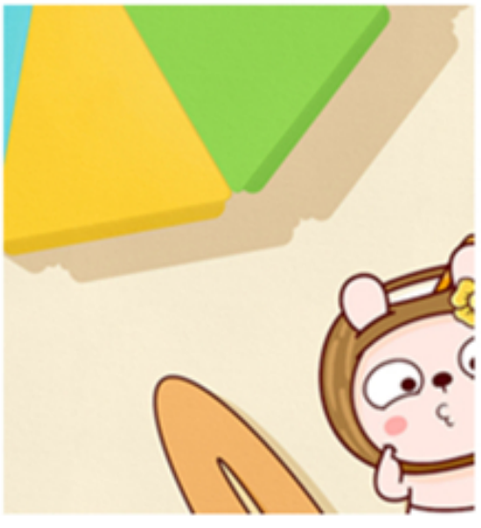
最新IT书籍PDF
最新IT书籍PDF
最新IT书籍PDF
最新IT书籍PDF



版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

图数据库技术丛书
张 帜 主编

[法] 杰罗姆·巴同 (Jérôme Baton)
[英] 里克·范·布鲁根 (Rik Van Bruggen) / 著
庞国明 胡佳辉 苏 亮
晏永年 于松林 李建潮 / 译

Neo4j 3.x 入门经典

Neo4j数据建模、性能调优和数据可视化技术

清華大學出版社

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

翻译团队介绍

张 帜（张老师）

全书总负责人，图数据库技术丛书主编。

张老师是微云数聚（北京）科技有限公司创始人、中国IT界元老、中国图数据库先导者、大数据领域资深专家、WPS曲线汉字全套核心技术的发明人。1985年研究生毕业于国防科技大学，获中国首届信息系统工程硕士。曾长期在微软任高级软件设计师及"维纳斯计划"技术主管。曾牵头研发中国移动139手机邮箱等明星商业产品。于2017年两会期间做客CCTV证券资讯频道《超越》栏目，与著名主持人阿丘老师对话，畅谈《关于图数据库的梦想》，被誉为中国图数据库第一人。推广普及图数据库并研发自主可控的国产图数据库，将是他人生的下一个关键目标！

主要著作有：

- 清华大学出版社2017年出版的《Neo4j权威指南》。
- 电子工业出版社1993年出版的《Turbo Pascal大全》，再版3次。
- 湖南科学技术出版社1988年出版的《IBM PC TURBO PASCAL程序设计手册》，再版2次。
- 湖南科学技术出版社1981年出版的《数据分析中的典型例题和解题方法》，全国大学数学专业推荐参考书，再版8次。

张老师创办的微云数聚（北京）科技有限公司是一家实力雄厚的大数据技术公司，现有三个团队，分别是大数据专家团队、移动互联网技术团队和图数据库技术团队。微云数聚专注于研究图数据库技术及其应用，是世界领先的图数据库Neo4j的战略合作伙伴和在中国的官方代理。公司研制的Neo4j 简体中文版是专为中国企业量身打造、符合中国企业习惯的图数据库系统产品，除了提供系统的汉化之外，还能扩展支持节点的图片显示、数据驱动显示呈现、智能查询及导入精灵（一种支持简便导入Excel、MySQL和Oracle等数据的工具），这些扩展将极大地促进Neo4j在华语地区的推广使用。微云数聚在华为、CCTV和中国首席数据官联盟等的大力支持下，与中国互联网、大数据企业客户建立了良好的合作和信任关系，为Neo4j的市场开拓奠定了良好的基础。

严禁网络传播本PDF！ 版权所有，严禁传播！

非卖品，仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用
严禁网络传播本PDF，违者责任自负！
版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

翻译团队介绍

庞国明（小 明）

负责全书翻译工作管理，完成第2、11、13章的翻译。

小明是《Neo4j权威指南》副主编、Neo4j中文社区创始人、系统架构师、高级信息系统项目管理师；组织并参与“智能公共交通网络”“多社交平台人脉网络”等多个Neo4j应用项目的研发，具有丰富的实战经验；自2008年以来，在《电脑编程技巧与维护》等国家级科技期刊发表近40篇文章。

胡佳辉（家 辉）

负责完成第4、6、7章的翻译。

家辉是西南交通大学计算机硕士，成都数之联科技有限公司软件架构师，《Neo4j权威指南》副主编；近年来主要从事大数据及相关系统的软件架构设计、机器学习、图数据库及区块链方面的研究。

严禁网络传播本PDF！
版权所有，严禁传播！
非卖品，仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用
严禁网络传播本PDF，违者责任自负！
版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

苏 亮（小 亮）

负责完成第1、9章的翻译。

小亮是《Neo4j权威指南》副主编、国防科技大学计算机博士，主要关注分布式计算、数据挖掘、数据库及大数据分析与处理，近年来从事行业大数据、图数据库及数据安全等方向的研发工作。

晏永年（永 年）

负责完成第8、10、12章和附录的翻译。

晏永年，图数据库及大数据领域专家，对Neo4j有着深入研究，长期从事百亿级数据规模的Neo4j图数据库的结构设计、性能优化。在国内公安等行业的海量数据处理与分析领域颇有建树，尤其是针对各类海量的人员行为所构建的关系进行最优化结构设计、高性能检索、海量异构库高性能数据同步等方面有着丰富的实战经验。

严禁网络传播本PDF！
版权所有，严禁传播！
非卖品，仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

翻译团队介绍

于松林（松 林）

负责完成第3、15章的翻译。

于松林，微云数聚（北京）科技有限公司联合创始人，北京理工大学计算机专业硕士，图数据库专家，Neo4j Technology Certification国际认证专家，对大数据、信息安全及移动互联网拥有丰富经验。

李建潮（建 潮）

负责完成第5、14章的翻译。

李建潮，现任设计IN技术负责人，10多年IT领域工作经验，国内敏捷开发先行者之一，曾任丁香园技术总监、钢为网CTO、纳斯达克上市公司Perficient首席架构师。

Neo4j 3.x

严禁网络传播本PDF！

版权所有，严禁传播！

非卖品，仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

图数据库技术丛书

张 帜 / 主编

Neo4j 3.x 入门经典

[法] 杰罗姆·巴同（Jérôme Baton）
[英] 里克·范·布鲁根（Rik Van Bruggen） / 著

庞国明 胡佳辉 苏 亮
晏永年 于松林 李建潮 / 译

严禁网络传播本PDF！ 版权所有，严禁传播！

非卖品，仅供非商业用途或交流学习使用

清华大学出版社

北 京

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

内 容 简 介

Neo4j 是一个高性能的 NoSQL 图数据库，将结构化数据存储在拓扑图上而不是表中，从而轻松地管理巨量的数据。Neo4j 凭借嵌入式、高性能、轻量级等优势越来越受关注。

本书共 15 章，介绍图论和图数据库、Neo4j 基础、Neo4j 数据模型、Cypher 入门、APOC 和 Cypher 扩展、查询性能优化、Neo4j 数据导入、空间、安全认证、Neo4j 可视化、Neo4j 数据重构、集群等内容，最后通过两大案例（推荐系统和影响分析与模拟）的讲解，使你获得图数据建模与设计的技能。

本书既可以作为 Neo4j 初学者的入门教材，也可以作为相关行业 Neo4j 技术专家的参考手册。

本书为 Packt 出版有限公司授权出版发行的中文简体字版本
北京市版权局著作权合同登记号　图字：01-2017-8959

本书封面贴有清华大学出版社防伪标签，无标签者不得销售
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

Neo4j 3.x 入门经典 /（法）杰罗姆·巴同，（英）里克·范·布鲁根著；张帜主编；庞国明等译.
— 北京：清华大学出版社，2019
（图数据库技术丛书）
书名原文：Learning Neo4j 3.x - Second Edition
ISBN 978-7-302-51918-8

I. ①N… II. ①杰… ②里… ③张… ④庞… III. ①关系数据库系统 IV. ①TP311.132.3

中国版本图书馆 CIP 数据核字(2018)第 288634 号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：李红英

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：三河市金元印装有限公司
经　　销：全国新华书店
开　　本：190mm×260mm　　彩　　插：2 页　　印　　张：13.75　　字　　数：352 千字
版　　次：2019 年 2 月第 1 版　　印　　次：2019 年 2 月第 1 次印刷
定　　价：69.00 元

产品编号：078318-01

严禁网络传播本PDF！　版权所有，严禁传播！

非卖品，仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

Neo4j 3.x 入门经典

第 2 版

Copyright © 2017 Packt Publishing

版权所有，未经出版者事先书面许可，不得将本书的任何部分复制、存储在检索系统中，也不得以任何形式或任何方式进行传播，除非嵌入关于文章或评论中的引语。

为了确保所提供信息的准确性，本书在编写过程中已经尽了一切努力。但是，本书并不能完全担保所有信息都是正确的。作者、Packt 出版公司及其经销商和分销商都不对本书直接或间接引起或声称造成的任何损害承担责任。

Packt 出版社努力通过适当方式提供有关本书中提到的所有公司和产品的商标信息。但是，Packt 出版社不能保证这些信息的准确性。

首次出版：2014 年 8 月

第 2 版：2017 年 10 月

生产参考：1171017

由 Packt 出版社出版

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78646-614-3

www.packtpub.com

严禁网络传播本PDF！ 版权所有，严禁传播！

非卖品，仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

参与人员

作者：Jérôme Baton、 Rik Van Bruggen
印刷编辑：Tasneem Fatehi
审稿人：Taffy Brecknock、 Jose Ernesto Echeverria 、Adriano Longo
项目协调员：Manthan Patel
调整编辑：Amey Varangaonkar
校验：Safis Editing
材料编辑：Vinay Argekar
索引编辑：Tejal Daruwale Soni
内容编辑：Jagruti Babaria 、Tejas Limkar
图片整理：Tania Dutta
技术支持：Dinesh Chaudhary、Dharmendra Yadav
出版协调员：Deepika Naik

严禁网络传播本PDF！ 版权所有，严禁传播！

非卖品，仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

关于作者

JérômeBaton（杰罗姆·巴同）起初从事黑客攻击计算机，首先从攻击游戏开始，之后通过在 Amstrad CPC 上自学了 Basic，开始编写一个程序扰乱视频控制器的运行。

然后，为了游戏的原因，他转而使用了 Commodore Amiga 和 AMOS Basic。后来通过认真学习，他在大学的个人计算机和大型机上编写了 Turbo Pascal、C、COBOL、Visual C ++和 Java，甚至在高中编写了 Logo。然后，Java 问世，他成了一个顾问，主要从事编写许多不同业务网站的后端代码。

Jérôme 在 Neo4j、JBoss Forge、Devoxx4Kids 的 Arduino 研讨会上发布了多篇文章，并参与评估了 Android 的几本书。

Jérôme 很喜欢在法国的大学任教，包括巴黎大学 V-RenéDescartes（Neo4j，Android）和特鲁瓦大学（Neo4j），他努力帮助学生学习。

如果你是一个有追求的人，那么在你生活中应至少有一次对所有事物产生怀疑。

Rene Descartes

（选自 https://www.brainyquote.com/authors/rene_descartes。）

在不编程的时候，Jérôme 喜欢摄影、做电子产品和一切 DIY，以及理解事物的运作方式，并试图在 Twitter 上变得聪明或有趣，花费大量的时间来理解他的孩子和感悟生活。

Rik Van Bruggen（里克·范·布鲁克）是 Neo4j 在英国 Benelux 经济联盟和北欧地区的销售副总裁。他一直在创业公司工作，包括 eCom Interactive Expertise、SilverStream Software、Imprivata 和 Courion。虽然他对技术有兴趣，但他真正的热情在于业务以及如何使技术为业务工作。他和妻子以及三个可爱的孩子住在比利时的 Antwerp，他们喜欢技术、野外活动、跑步和比利时啤酒。

严禁网络传播本PDF！ 版权所有，严禁传播！

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

版权相关注意事项：

1、书籍版权归著者和出版社所有

2、本PDF来自于各个广泛的信息平台，经过整理而成

3、本PDF仅限用于非商业用途或者个人交流研究学习使用

4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负

5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF

6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅

7、请于下载PDF后24小时内研究使用并删掉本PDF

鸣　谢

严禁网络传播本PDF！ 版权所有，严禁传播！

对于这个项目，我要感谢很多人，这对我来说确实是一个伟大的个人成就。

首先，Rik Van Bruggen 是这本书的原作者，也就是说我是站在巨人的肩膀上完成此书的。其次，Packt 出版社的 Vinay 和 Jagruti 为我提供了很大的帮助。

感谢 Neo4j 的 William LyOn、CédricFauvet、Mark Needham、BenOit Simard、Michael Hunger、Craig Taverner 和 Jim Webber 在过去几年的帮助，并感谢他们从 Stack Overflow、Slack 分享他们的知识。

感谢巴黎 IUT 大学的 Daniel'DG Guillaume、FrançoiseMeunier、Florence Fessy-Mesatfa 和 JérômeFessy 这样鼓舞人心的老师，还有 Robert T Hughes 博士、Richard N Griffith 和布赖顿大学的 Graham Winstanley。

在过去，有很多老师激励我分享知识，我仍然记得他们：Legrand 夫人、Viala 夫人和 Bouhadda 先生。另外，还有不以英语为母语的人，我起初英语讲得很不好，多亏他们的帮助：Goddard 夫人和 Maluski 夫人对我学习第二语言帮助很大。

知识改变命运！

还要感谢我的医生：Vive laSécuritéSociale，没有他估计我已经成为纪念品了！

基本上，我要感谢所有我从中学到现在的所有人，无论是老师还是同学，也包括我的学生。

谢谢你，Romin Irani（@iRomin），我的朋友，你是一个好榜样。

谢谢 AnnyNaïm，你是一个真正的光辉人物。

最重要的是，爱你，kiddos！

我真的应该画出所有我想感谢的人。

非卖品，仅供非商业用途或交流学习使用

关于本书审稿人

Taffy Brecknock 在 IT 行业从业 20 多年。在他的职业生涯中，他曾担任软件开发人员、开发团队管理人员，并负责应用程序设计和系统架构。

他曾担任政府和私营机构的组织人员。在与澳大利亚政府合作的过程中，Taffy 参与了政府执法部门的相关业务。在很多项目中他都使用传统的关系型数据库，并且经常遇到一些难以解决的困难。

在了解图数据库尤其是 Neo4j 之后，他对这项技术的特殊应用变得非常感兴趣。他认为，在当今的商业世界中，很多复杂关系都可以通过图结构来处理。

Jose Ernesto Echeverria 于 20 世纪 90 年代开始从事关系数据库相关工作，从 2014 年起一直从事 Neo4j 的相关工作。他很善于数据建模，并且更喜欢图数据库。作为一名多语种编程人员，他使用 Java、Ruby 和 R 等多种开发语言来操作 Neo4j 以解决跨国公司的数据管理问题。他是 GraphConnect、OSCON 和 RailsConf 组织的参与者。不工作时，他喜欢陪伴家人、公路旅行、带孩子以及阅读和饮酒。

Adriano Longo 是一位荷兰的数据分析师，对 Neo4j 的数据模型充满热情。

他擅长用 Cypher、R、Python 和 SQL 查询数据、数据建模。之前曾在 UEA 气候研究部门从事气候预测模型的相关工作，然后专注于私营企业的分析解决方案。

如今，Adriano 使用 Neo4j 和 Linkurious.js 来探索复杂的关系网络。

严禁网络传播本PDF！ 版权所有，严禁传播！
非卖品，仅供非商业用途或交流学习使用

www.packtpub.com

请访问 www.PacktPub.com 来获得本书相关的支持和文件下载。Packt 出版社为每一本书提供电子版本（PDF 和 ePub 文件）。你可以通过 www.PacktPub.com 获取电子版本，对于购买纸质书籍的用户你可以享有相应的折扣。更多详情请通过 service@packtpub.com 联系我们。另外，在 www.PacktPub.com 上，你还可以阅读一系列免费技术文章，注册订阅一系列免费电子报刊，并在 Packt 书籍和电子书上获得独家折扣和优惠。

Mapt

网址：http://www.packtpub.com/mapt

通过 Mapt 你可以获得最多的软件技能。通过 Mapt 你就可以浏览 Packt 的所有书籍和视频课程，另外我们还有行业领先的工具来促进你的个人发展、推动你的职业规划。

通过订阅可以：

- 检索 Packt 出版的每本书。
- 复制并粘贴、打印书签内容。
- 通过网页浏览器访问相关资源。

严禁网络传播本PDF！ 版权所有，严禁传播！
非卖品，仅供非商业用途或交流学习使用

读者反馈

严禁网络传播本PDF！

版权所有，严禁传播！

非卖品，仅供非商业用途或交流学习使用

感谢你购买这本书。在 Packt 出版社，质量是我们的核心。为了帮助我们改进本书的内容，请通过 https://www.amazon.com/dp/1786466147 登录 Amazon 并留下你的宝贵评论。

如果你想加入我们的审核团队，请通过电子邮件 customerreviews@packtpub.com 来与我们取得联系以成为我们的审稿人。我们会发放免费的电子书和视频给审稿人，以换取宝贵的反馈意见，帮助我们不懈地改进我们的产品！

译者序

如今 NoSQL 旗下大量新技术不断涌现，传统关系数据库的霸主地位已经岌岌可危。Neo4j 作为 NoSQL 类下的图数据库为处理数据之间的复杂关系提供了优越的解决方案。Neo4j，作为图数据库领域的领头羊，不断引领图数据库的发展。自 Neo4j 诞生以来，它被应用到社交、推荐引擎、电信、路由控制、金融反欺诈、地理空间等多个领域。

目前市面上有关图数据库的中文书籍屈指可数，随着图数据库在中国的推广和普及，大家迫切需要引入讲述 Neo4j 技术的权威书籍。在此情境下，走在国内图数据库领域最前沿的微云数聚创始人张帜老师带领由庞国明、胡佳辉、苏亮、晏永年、于松林、李建潮组成的团队共同翻译完成本书，参加本书翻译的译者都是来自相关领域的专家，有着深厚的应用和技术背景，保证了本书的翻译质量。

另外，微云数聚的张帜老师出版了 Neo4j 领域的一系列书籍，其中包括《Neo4j 权威指南》（一本图数据库专业的书，详细介绍了 Neo4j 的各个技术点）。在不久的将来，《APOC 指南》将会出版，它将专门介绍 Neo4j 数据库的 APOC 技术，请大家拭目以待。

本书由英文书翻译而来，难免会有欠妥之处，恳请广大读者不吝指正（可以将你的宝贵建议发表在 Neo4j 中文社区（neo4j.com.cn）或通过邮件发送到 neo4j@we-yun.com）。

严禁网络传播本PDF！ 版权所有，严禁传播！
非卖品，仅供非商业用途或交流学习使用

前言

本书内容基于 Neo4j 3.x 版本，通过对本书的学习我们会掌握图数据库和 Neo4j 的关键技能。从理论知识到应用程序开发，我们将全方面地学到 Neo4j 的各项知识。

本书所涵盖的内容

第 1 章 图论和数据库，解释图数据库技术的历史和基本理论。另外，本章阐述图数据库不断的发展、演变，以及图数据库从技术、行业方面与其他数据库技术进行比较。

第 2 章 Neo4j 基础入门，介绍图数据库 Neo4j 的具体实现，并着重介绍 Neo4j 的关键概念和特性。

第 3 章 Neo4j 数据模型，阐述图数据库的基本建模技术。

第 4 章 Cypher 入门，阐述 Cypher 查询语言的用法。

第 5 章 过程——APOC，在本章我们将学习如何在 Cypher 查询中使用过程、获取过程的相关信息以及社区版提供的数百个存储过程的概述。

第 6 章 Cypher 扩展，阐述如何扩展 Neo4j 实例的功能、如何创建用户自定义 APOC。

第 7 章 查询性能优化，阐述如何调整我们的 Cypher 查询以获得更好的性能。

第 8 章 Neo4j 数据导入，阐述如何从不同数据源将数据导入到 Neo4j。

第 9 章 空间，阐述如何使用 Neo4j 的地理信息功能。

第 10 章 安全认证，阐述 Neo4j 中的认证和授权。

第 11 章 Neo4j 可视化，阐述如何展示我们的数据。

第 12 章 Neo4j 数据重构，阐述如何按照需求来修改适合我们的数据模型。

第 13 章 集群，使用 Neo4j 企业版建立一个因果集群。

第 14 章 案例——推荐系统，深入挖掘特定的图数据库用例，并使用特定的示例数据集和查询模式进行剖析。

第 15 章 案例——影响分析与模拟，分析网络中某一部分的变化对另外一部分的影响。

附录 应用技巧介绍，阐述使用 Neo4j 的一些技巧。

学习本书需要具备的条件

要运行 Neo4j 的软件，我们需要一个装有 Java7 或更高版本的计算机，并且要具备 4GB 的内存和 2GB 的可用硬盘空间。

严禁网络传播本PDF！ 版权所有，严禁传播！
非卖品，仅供非商业用途或交流学习使用

本书有一些在 GNU / Linux 系统运行的例子，所以某些内容会用到 Linux 操作系统。

除了第 10 章安全认证和第 13 章集群之外，大多数章节讨论的内容都适用于 Neo4j 社区版和 Neo4j 企业版。

在最后几章内容中，我们推荐使用两台笔记本电脑，几个树莓派设备和 Docker 容器来做

实验。

本书适用人群

本书内容适用于想要采用图作为数据存储结构或者想要处理复杂网状关系数据的开发人员。只要我们有传统数据库的使用经验，即便没有图数据库的相关经验我们也是可以很容易地理解本书内容。

本书内容格式的约定

代码块样式如下：

```
public class StringHacking {
@UserFunction
 @Description("Returns the last word of a string")
 public String getLastWord( @Name("aStr") String aStr){
 if(aStr==null) return null;
 else {
 int pos = aStr.lastIndexOf(" ");
 if(pos==-1) return aStr;
 else return aStr.substring(pos+1);
 }
 }
}
```

当本书想要让读者着重注意代码中某一部分时，将会加粗显示：

```
MATCH (ln:LastName)--(p:Person)
RETURN ln, learningneo4j.randomCount(p.lastName) AS badcount
```

命令行的输入、输出内容格式如下：

```
mkdir data/ldap/environment -p
mkdir data/ldap/db -p
```

新术语和**重要词汇**以及操作界面上的词汇（如菜单、对话框的名字）本书将以粗体显示，如下所示：

为了下载新模块，我们打开“**文件|设置|项目名称|项目解释器**”。

严禁网络传播本PDF！ 版权所有，严禁传播！
非卖品，仅供非商业用途或交流学习使用

	重要提醒的内容将以此形式展示。
	技巧性的内容将以此形式展示。

反馈

我们欢迎读者反馈对本书的看法，任何你喜欢或不喜欢的内容都可以反馈给我们。读者的反馈对我们是非常重要的，从读者的反馈中我们可以知道如何修改、完善本书的内容。如果要向我们反馈关于本书的普通问题请发送电子邮件至 feedback@packtpub.com，并在邮件主题中写明章节标题；如果你是一位专业人士并希望参与到本书的撰写中，请参阅我们的作者指南，网址为 www.packtpub.com/authors。

读者支持

既然你是这本书的拥有者，我们自然愿意从多方面帮助你从购买中获得最大收益。

下载本书示例代码

如果你是从 http://www.packtpub.com 上购买的本书，那么你可以直接在此网站下载本书的示例代码文件。如果你在其他地方购买了本书，则可以访问 http://www.packtpub.com/support 并注册，我们将直接将示例代码文件通过电子邮件发送给你。你可以通过以下步骤下载代码文件：

（1）使用你的电子邮件地址和密码登录或注册我们的网站。
（2）将鼠标指针悬停在顶部的 SUPPORT 选项卡上。
（3）点击 Code Downloads&Errata。
（4）在搜索框中输入书名。
（5）选择你要下载代码的对应书名。
（6）打开购买本书的下拉菜单。
（7）点击代码下载。

示例代码文件是一个压缩包文件，当你下载下来后请使用最新版本的解压工具，如：

- WinRAR、7-Zip Windows 版。
- Zipeg 、iZip、UnRarX Mac 版。
- 7-Zip 、PeaZip Linux 版。

从 GitHub 上也可以下载到本书的示例代码文件，GitHub 地址为 https://github.com/PacktPublishing/Learning-Neo4j-3x-Second-Editon。我们还在 https://github.com/PacktPublishing/上提供了丰富而有用的代码。

严禁网络传播本PDF！ 版权所有，严禁传播！
非卖品，仅供非商业用途或交流学习使用

版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

目 录

第 1 章 图论和数据库 1
1.1 Neo4j 3.x 简介和图的历史 1
1.2 图论的定义和用途 3
1.2.1 社会学研究 5
1.2.2 生物学研究 6
1.2.3 计算机科学 6
1.2.4 流量问题 7
1.2.5 路径问题 8
1.2.6 网页搜索 9
1.3 背景 10
1.3.1 导航数据库 11
1.3.2 关系数据库 13
1.3.3 NoSQL 数据库 15
1.4 图数据库的属性图模型 18
1.4.1 节点标签 20
1.4.2 关系类型 20
1.5 使用图数据库的注意事项 20
1.5.1 为什么使用图数据库 20
1.5.2 什么时候不用图数据库以及用什么替代 22
1.6 问答 23
1.7 小结 23

第 2 章 Neo4j 基础入门 25
2.1 Neo4j 的关键概念和特点 25
2.1.1 从头开始创建图 25
2.1.2 基于事务的 ACID 数据库 26
2.1.3 可用于联机事务处理 27
2.1.4 扩展性能 28
2.1.5 声明式查询语言 Cypher 28
2.2 Neo4j 典型案例 29
2.2.1 复杂的连接密集型查询 29
2.2.2 路径查询 29
2.2.3 开源解决方案 29

2.3 特性 30
2.4 许可协议 31
2.5 安装 Neo4j 32
2.5.1 在 Windows 系统上安装 Neo4j 32
2.5.2 在 Mac 和 Linux 系统上安装 Neo4j 35
2.6 在云平台使用 Neo4j 37
2.7 沙盒模式 41
2.8 在 Docker 容器中使用 Neo4j 47
2.8.1 安装 Docker 48
2.8.2 准备文件 48
2.8.3 在 Docker 容器中运行 Neo4j 48
2.9 问答 50
2.10 小结 51

第 3 章 Neo4j 数据模型 52
3.1 四种基础数据结构 52
3.2 图数据库建模基础 53
3.2.1 ER 图和关系模式 53
3.2.2 通过连接表引入复杂性 54
3.3 图模型——一个简单、高保真的现实模型 55
3.4 图建模——最佳实践和陷阱 57
3.4.1 图建模最佳实践 57
3.4.2 图数据库建模陷阱 61
3.5 问答 64
3.6 小结 65

第 4 章 Cypher 入门 66
4.1 Cypher 的语法 66
4.2 Cypher 的关键特性 67
4.3 与数据很好地结合 68
4.3.1 创建数据 68
4.3.2 读取数据 69
4.3.3 更新数据 70
4.3.4 删除数据 71
4.4 Cypher 的关键操作词 72
4.5 语法约定 73
4.6 高级知识 74
4.7 Cypher 参考卡 78
4.8 openCypher 工程 79
4.9 小结 79

第 5 章 过程——APOC 80
5.1 安装 APOC 80
5.1.1 机器安装 81
5.1.2 Docker 容器安装 81
5.2 检查 APOC 安装 81
5.3 函数和过程 82
5.4 习惯用法 82
5.4.1 来自朋友的建议 82
5.4.2 图概述 83
5.5 关键用法 85
5.5.1 安装 85
5.5.2 随机图生成器 85
5.5.3 网站排名 86
5.5.4 Cypher 语句限时执行 86
5.5.5 节点集合连接 87
5.5.6 更多 APOC 87
5.6 问答 88
5.7 小结 89

第 6 章 Cypher 扩展 90
6.1 构建扩展工程 90
6.1.1 创建函数 91
6.1.2 创建过程 92
6.2 定制聚合器 95
6.3 非托管扩展 96
6.3.1 HTTP 和 JAX-RS refreshers 97
6.3.2 JSON 响应流式化 98
6.4 小结 101

第 7 章 查询性能优化 102
7.1 Explain 和 Profile 指令 102
7.1.1 查询计划 102
7.1.2 运算符 104
7.2 索引 105
7.2.1 强制使用索引 105
7.2.2 强制使用标签 105
7.3 经验法则 106
7.3.1 Explain 所有查询 106
7.3.2 行 106
7.3.3 不要过度消耗资源 106
7.3.4 返回笛卡儿积级别的数据 106

7.3.5 简单 106
7.4 小结 107

第 8 章 Neo4j 数据导入 108
8.1 导入 CSV 文件 108
8.2 导入 JSON 源 111
8.3 导入 JDBC 源 114
8.3.1 测试安装 114
8.3.2 导入所有系统数据 114
8.4 导入 XML 源 115
8.5 小结 116

第 9 章 空间 118
9.1 空间简介 118
9.1.1 回顾 118
9.1.2 无缺陷塔 119
9.1.3 什么是空间 119
9.2 Neo4j 的空间特性 119
9.3 APOC 的空间特性 120
9.4 地理编码 121
9.4.1 设置 OSM 为提供商 121
9.4.2 设置谷歌为提供商 121
9.5 Neo4j Spatial 122
9.5.1 在线示例 122
9.5.2 Neo4j Spatial 特性 122
9.6 导入 OpenStreetMap 数据 124
9.7 大型 OSM 数据导入 126
9.7.1 简易方法 126
9.7.2 更直接的数据导入方法 127
9.8 查询卫生间 127
9.9 理解 WKT 和 BBOX 128
9.10 删除所有地理数据 128
9.11 小结 129

第 10 章 安全认证 130
10.1 认证与授权 130
10.2 角色 131
10.3 用户管理 132
10.4 Neo4j 连接到 LDAP 目录 132
10.5 使用 LDAP 配置 Neo4j 135
10.6 问答 136
10.7 小结 136

第 11 章 Neo4j 可视化 137
11.1 可视化的重要作用 137
11.1.1 为什么图数据可视化很重要 137
11.1.2 直观地与数据进行交互 137
11.1.3 模式识别 138
11.1.4 发现重点 139
11.2 图数据可视化的基本原则 139
11.2.1 开源可视化组件库 140
11.2.2 将可视化组件库应用到项目中 143
11.2.3 可视化解决方案 144
11.2.4 问题和陷阱 148
11.2.5 Cytoscape 实例 149
11.3 问答 152
11.4 小结 153

第 12 章 Neo4j 数据重构 154
12.1 预备步骤 154
12.2 简单改变 154
12.2.1 重命名 154
12.2.2 添加数据 155
12.2.3 删除数据 156
12.3 重大改变 156
12.3.1 知道我们的模型 156
12.3.2 重构工具 157
12.3.3 属性变标签 157
12.3.4 属性变节点 157
12.3.5 相关节点变标签 158
12.3.6 合并节点 158
12.3.7 关系 159
12.4 结论 162
12.5 小结 162

第 13 章 集群 163
13.1 为什么需要 Neo4j 集群 163
13.2 Neo4j 集群的概念 164
13.2.1 核心服务器 164
13.2.2 只读副本服务器 164
13.2.3 高吞吐量 164
13.2.4 数据冗余 165
13.2.5 高可用 165
13.2.6 Bolt 协议 165

13.3 构建 Neo4j 集群 165
13.3.1 核心服务器 165
13.3.2 只读副本服务器 167
13.3.3 bolt+routing 协议 168
13.4 在 Neo4j 集群中实现灾难恢复 169
13.5 小结 169

第 14 章 案例——推荐系统 170
14.1 推荐系统剖析 170
14.2 推荐系统的图模型应用 171
14.3 推荐系统查询示例 173
14.3.1 基于商品购买的推荐 173
14.3.2 基于品牌忠诚度的推荐 175
14.3.3 基于社交关系的推荐 176
14.3.4 组合推荐 176
14.4 推荐系统的业务变化 177
14.5 欺诈检测系统 178
14.6 访问控制系统 179
14.7 社交网络系统 179
14.8 问答 180
14.9 小结 181

第 15 章 案例——影响分析与模拟 182
15.1 影响分析系统解析 182
15.1.1 影响分析在业务流程管理中的应用 183
15.1.2 业务图建模 183
15.2 成本计算环境中的影响模拟 187
15.2.1 产品层次结构的图建模 187
15.2.2 产品层次结构图的应用 188
15.3 问答 192
15.4 小结 193

附录 应用技巧介绍 194

第 1 章

图论和数据库

学习方法因人而异。众所周知，背景知识将有助于更好地学习。这就是为什么在本书的开篇先介绍一些背景知识，它不是讲述历史故事，而是一个必要的背景展示，以便更好地抓住主题。

为此，我们将探讨下列话题：

- 图（Graph）：什么是图，从何而来。此节将明确本书的主旨，即包含哪些内容、不包含哪些内容。
- 图论（Graph theory）：什么是图论，其用途如何。本节将介绍一些图论应用示例，并为图数据库（例如：Neo4j 等）的产生做铺垫。
- 数据库（Database）：介绍不同种类的数据库及其用途。这有助于我们为不同的项目选择最合适类别的数据库。

接下来让我们深入研究吧。

1.1 Neo4j 3.x 简介和图的历史

在生活中经常用到“图（Graph）”这个词。然而，其含义很有可能与本书不同。事实上，大多数人（显然不是你，否则你可能不会拿起这本书。）当谈论到图时会想到：饼图（pie charts）、柱状图（bar charts）、图形（graphics）等，这些都不是我们所说的图。

本书将会遇到一个完全不同的课题——图，它很可能在数学课上出现过。我清楚地记得，在大学的离散数学课堂上，发现它非常复杂、很难懂。难以想象我后来的职业生涯会在软件中使用离散数学的相关知识，更不用说我将写一本关于图这个主题的书了。

那图到底是什么呢？为了回答这个问题，有必要了解其相关的历史背景。图实际上是一个相当古老的概念，首次出现在著名的瑞士数学家莱昂哈德·欧拉（Leonhard Euler）的学术论文中。他试图解决一个古老的问题，即我们现在所说的哥尼斯堡七桥问题（Seven Bridges of Königsberg），该问题很容易理解。

哥尼斯堡当时是普鲁士帝国位于普雷格尔河上一个美丽的中世纪城市，现今属于俄罗斯，介于波兰和立陶宛之间。如果在现今地图上试着去查找它，很可能会找不到，因为它现在被称作**加里宁格勒**。普雷格尔河不仅把哥尼斯堡分为左右两岸，还在河中央形成了一个小岛，叫 Kneiphof、从而将城市分割为四块区域（记为 A、B、C 和 D），各区域由七座桥相

连（分别标为a、b、c、d、e、f和g），如图1-1所示。

- 这七座桥连接着城市的四个不同区域。
- 问题是游览这座城市，参观每一个区域，穿过每一座桥，仅此一次。

图1-1是欧拉在1736年的论文中对该问题的阐述。

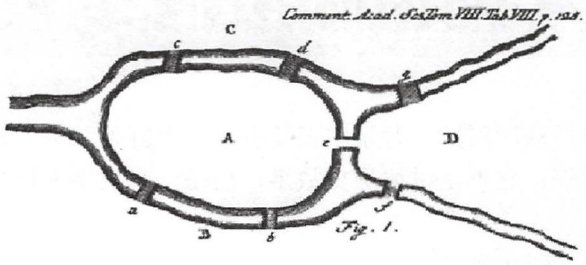


图1-1　1736年欧拉在其论文中对该问题的描述

本质上，这是一个寻径问题，类似的问题还有**骑士巡逻**问题（knight's ride problem）、**旅行商**问题（traveling salesman problem）等。现在根本称不上是一个非常困难的任务，然而，在当时，人们为试图解决此问题困惑了很长一段时间。直到欧拉的加入，对这个问题采取了一种非同寻常的数学方法，才得以彻底解决。

欧拉做了以下两件有趣的事：

- 首先，他并未采用传统的暴力搜索方法来解决这个问题（就是在地图上标出所有不同的可能路径，再试图找出是否有这样一条路线可穿越整个城市，暴力搜索的本质就是不断尝试、排查错误），而采用不同的方法。他以退为进，先对问题进行**抽象**，试图构建解决该问题领域的模型。至少在欧拉看来，大家的注意力集中到问题的错误部分——街道上。欧拉很快得出结论，哥尼斯堡的街道并没有真正解决问题。对于寻径，唯一重要的是以下几点：

  - 城市的各个区域
  - 连接城市各区域的桥

  至此，我们似乎突然发现了一个完全不同的问题，即“世界第一张图”，其准确的展现如图1-2所示。

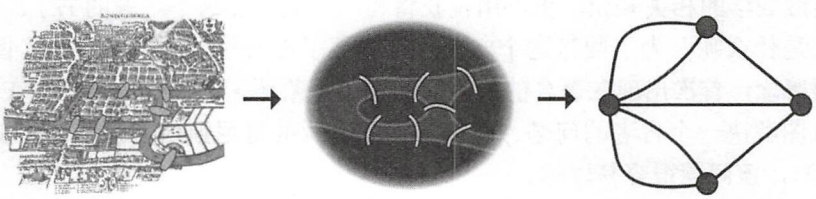


图1-2　简化的哥尼斯堡七桥问题

- 其次，欧拉将数学方法应用到他所创建的模型上，从而解决了这个难题。欧拉的逻辑很简单，如果想在哥尼斯堡城市漫步，就必须这样做：

- 必须从城市四个区域中的任何一个开始。
- 必须离开该区域，换句话说，必须穿过一座桥到城市的另一个区域。
- 然后要穿过另外 5 座桥，离开并进入城市的不同区域。
- 最后，将在城市的另一个区域结束。

因此，欧拉认为，第一个和最后一个区域必须有奇数座桥连接城市的其他区域（因为离开第一个区域、到达最后一区域仅一次），而其他两个区域必须有偶数座桥连接城市的第一个和最后一个区域，因为到达和离开同时出现。

连接城市区域桥的数量在欧拉创造的模型中具有非常特殊的意义，即模型的图表示，称之为图节点的度。为了使哥尼斯堡的每一座桥都经过一次，欧拉证明了所要做的就是运用一种非常简单的方法来建立城市各区域的度（换句话说就是计算桥的数量），如图 1-3 所示。

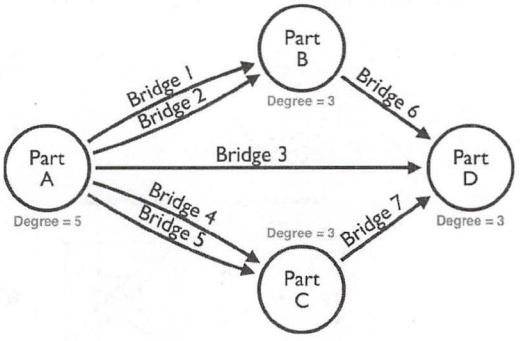


图 1-3　简化后的城市

这就是欧拉如何解决著名的“**哥尼斯堡七桥问题**”。通过证明城市的任何区域没有偶数座桥，从而证明了该漫游问题是不可能的。如果再增加一座桥，就立即使其成为可能，但在当时该城市和桥梁的状况下是无法做到“**欧拉行走**（Eulerian Walk）”的。

通过上述方法，欧拉创造了世界上第一张图。同样其研究技术具有普适性，任意一个图要能实现“欧拉行走”，必须仅有零个或两个奇度顶点，所有中间顶点度都必须为偶数。

总而言之，图只不过是两个或多个实体之间关联的抽象数学表示，为实体对象间的相互关系进行建模。通常由以下部分组成：

- 节点，即前面提到的对象或实体：数学中通常称为顶点（vertices），而在本书或图数据库中，比如 Neo4j，通常称为节点（nodes）。
- 节点之间的连接：数学中称为边（edges），而本书中称为关系（relationships）。
- 节点和关系相互连接构成一个图：许多重要的度量指标，例如连接到一个节点的边数（称之为度）就可以确定下来。许多其他度量指标也就自然形成。

既然已经讨论了图，还进一步认识了其性质和历史，现在就来看看在这些概念之上构建的规则，通常称之为图论。

1.2 图论的定义和用途

当欧拉发明第一张图时，他试图解决一个非常具体的问题，即哥尼斯堡七桥问题，有一个非常具体的表示或模型，及一个非常特定的算法。事实证明，还有相当多的问题，可归类如下：

- 用图来表征对象及其之间的关系。
- 在图上运用数学算法来求解。

该方法有相同之处，即研究上述问题建模和解决方法的学科，通常称为图论，是离散数学的一部分。

在这门学科中，有许多不同类型的图值得深入分析，如图 1-4 所示。

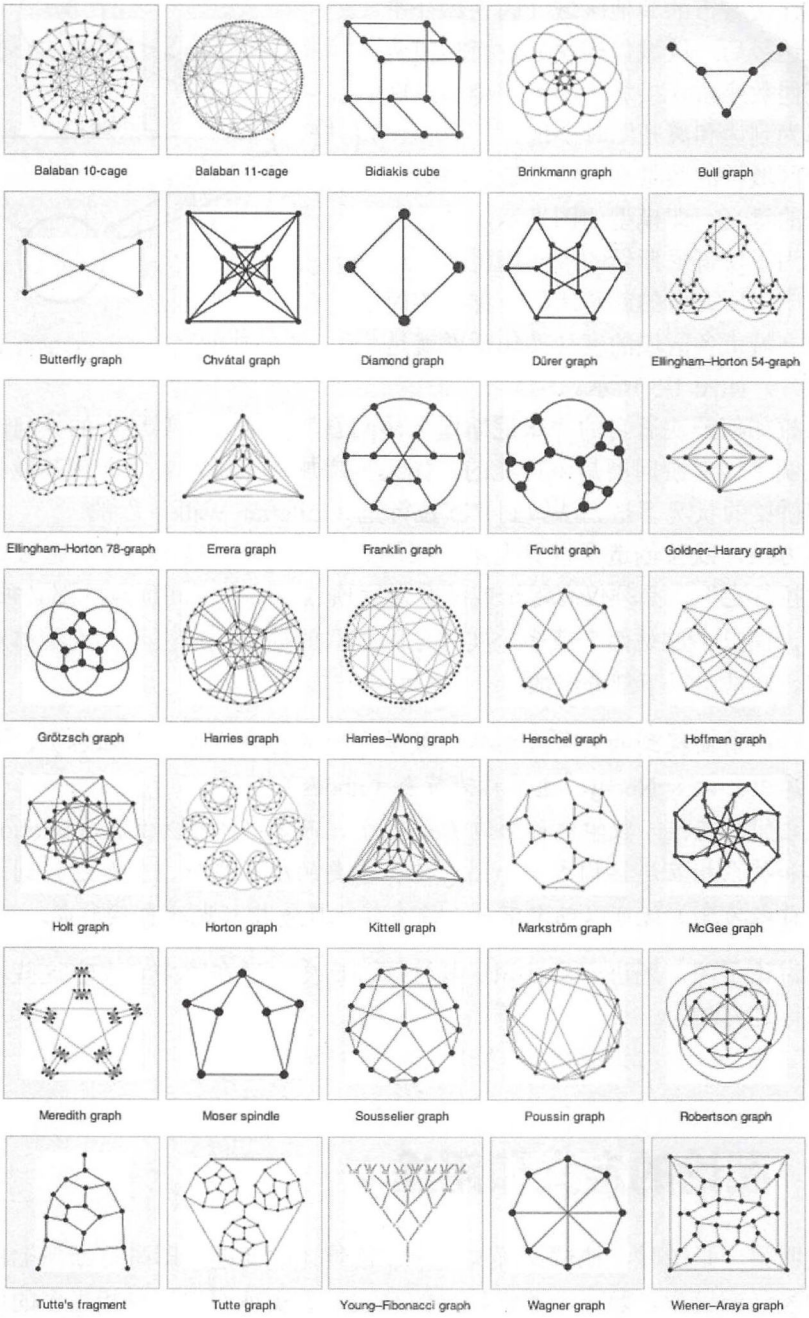


图 1-4　图的类型

图论，是研究图的建模和算法，已证明是一个迷人的、有着广泛应用前景的研究领域，同时解决了一些非常有意义的问题。有趣的是，它很少得到真正的应用；也许今天的科学家无须用到跨学科的方法（结合图论和其专业领域知识）。

因此，接下来将介绍一些图论相关的研究领域，但不是全部。相信这些例子将吸引大家继续阅读本书的后续章节，并对图数据库（如 Neo4j）产生浓厚的兴趣。

1.2.1　社会学研究

很早以前，人们就已经认知到人类相互作用的方式其实很容易通过网络方式来描述。每天人与人打交道，每天都在互相影响，每天都在交换想法。这些相互作用会通过他们所处的社会环境产生连锁效应。将这些相互作用用图来进行建模，能更好地理解全球人口结构、政治活动，以及特定群体对特定产品的商业价值等。随着在线社交网络的出现，这种以图为基础的社会认知方法已经有了一个全新的方向。像 Google、Facebook、Twitter、LinkedIn 等许多公司都已经做出了非常实际的努力，即基于图的应用系统，这些系统以客户或用户为目标，并从根本上改变了我们的日常生活。图 1-5 是我在领英（LinkedIn）网上的可视化呈现。

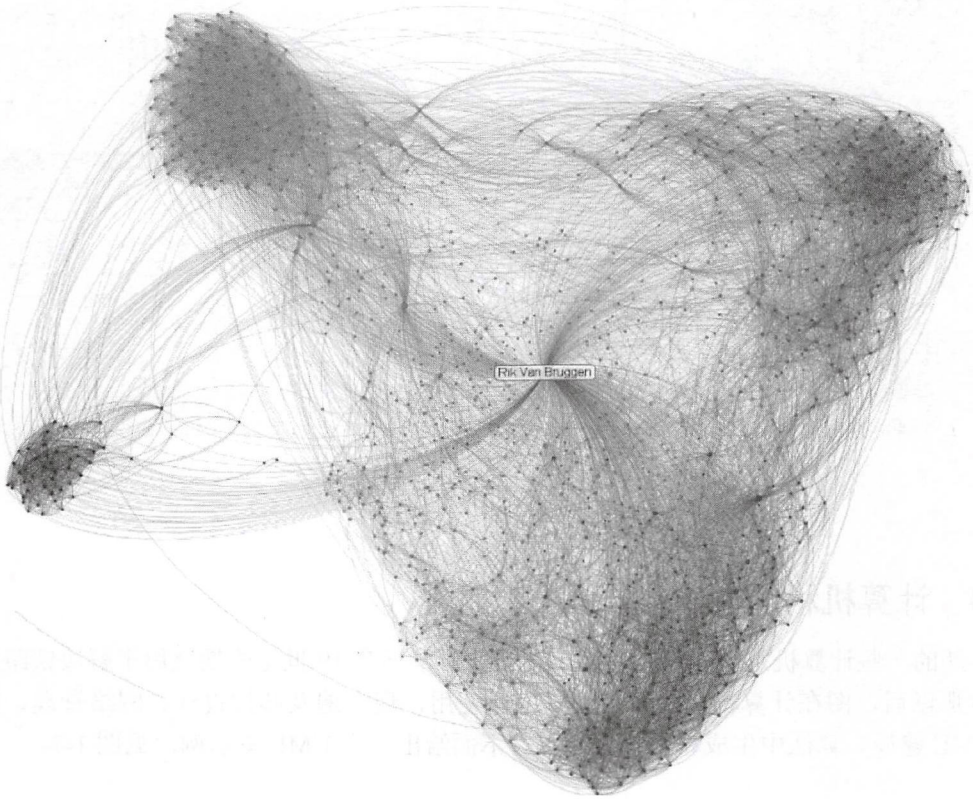


图 1-5　Rik 个人关系网的展现

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

1.2.2 生物学研究

常用的营销口号：图无处不在。这实际上是用一种非常真实而有趣的方式来刻画现实世界。同样，在生物学领域，研究人员很早就知道，生物成分（蛋白质、分子、基因等）以及它们的相互作用可以用图来精确建模和描述，这样将带来诸多现实好处。在代谢途径中（metabolic pathways，如图 1-6 所示的人体新陈代谢系统），图有助于了解人体不同部位的相互作用。在宏蛋白质组学（metaproteomics，从自然环境中提取蛋白质样本的研究）中，研究人员分析了不同种类蛋白质之间是如何相互作用的，以利于更好地控制化合或生物过程。

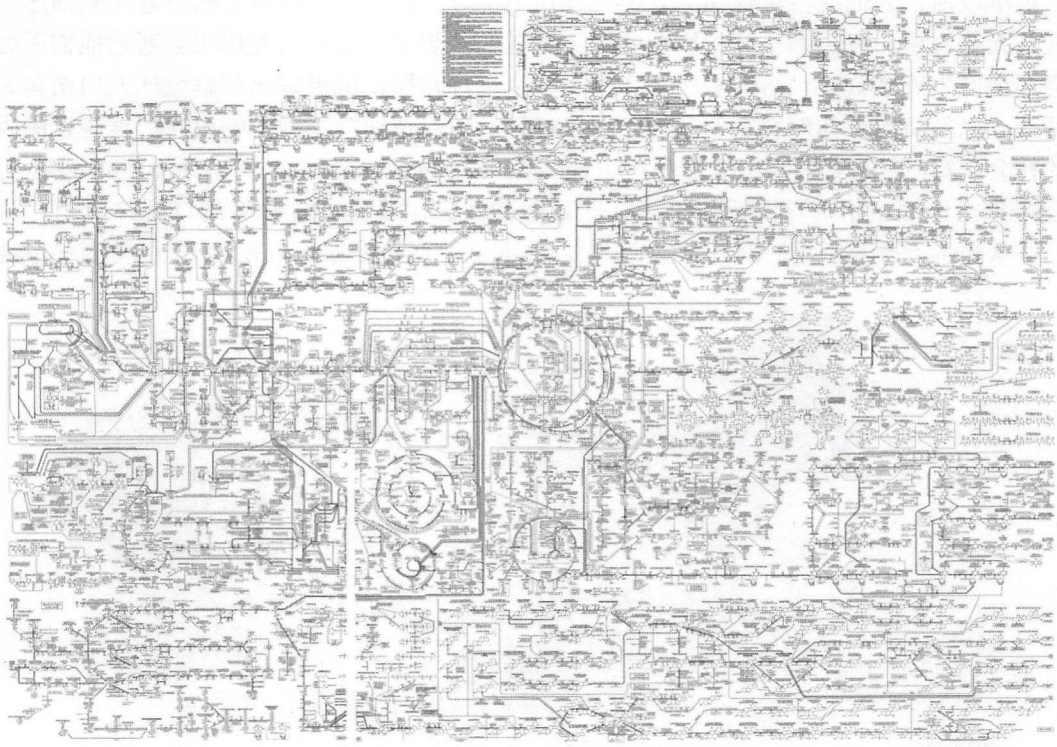


图 1-6 人类新陈代谢图

1.2.3 计算机科学

早期的一些计算机是用图来设计的。图计算引擎早在 19 世纪晚期就用于解决铁路调度问题，自那以后，图在计算机科学中得到了广泛使用，现今遍及芯片设计、网络管理、推荐系统、UML 建模、算法中生成和依赖分析等。下面给出一个 UML 图示例，见图 1-7。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

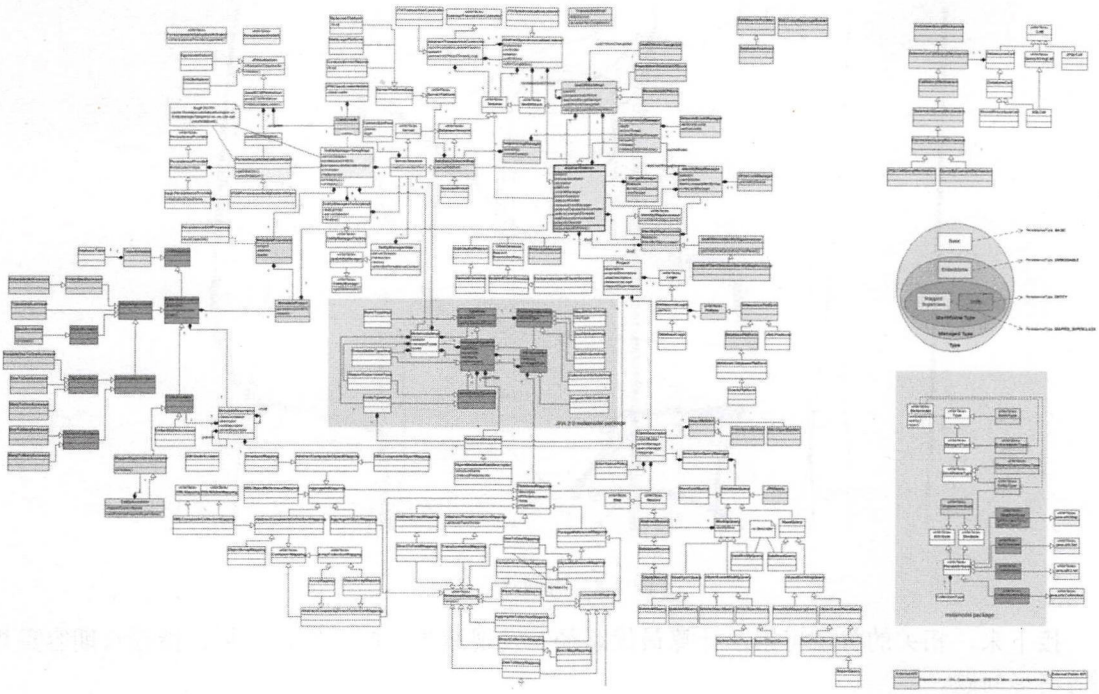


图 1-7　UML 图示例

还有一个有趣的案例是采用路径发现算法，软硬件工程师可以不间断地分析系统设计中的变化对系统其他部分带来的影响。如果源代码的一部分做了更改，例如，一个特定对象（object）被重命名，依赖分析算法可以轻松地遍历整个系统图，以找出其他受影响的类（class）。

1.2.4　流量问题

流量问题（flow problems），也称为最大流问题（maximum flow problems），是图论的又一个非常有趣的应用领域。事实上，从大的方面来讲它是一个优化问题，试图在整个流网络中找到最佳路径。流网络图中，节点（顶点）通过关系（边）相互连接，关系（边）代表特定容量。例如，在电信网络、天然气网络、航空网络、包裹递送网络等诸多领域案例中，通常在图模型的基础上结合复杂的算法。图 1-8 为 http://enipedia.tudelft.nl/网站上提供的类似流网络。

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

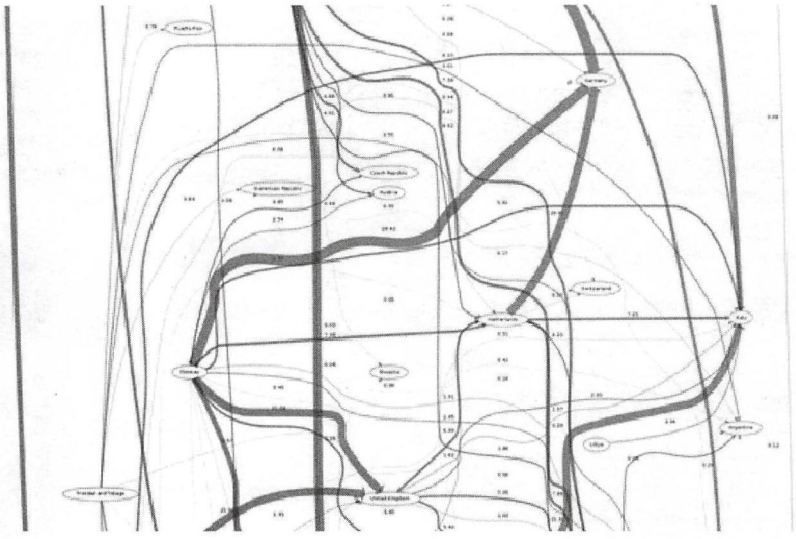


图 1-8　流网络示例

接下来，相关的算法可用于计算最优路径、发现瓶颈、制定维护计划、指导长期容量规划等诸多问题。

1.2.5　路径问题

之前提到的十八世纪欧拉解决的哥尼斯堡七桥问题，实际上是一个路径规划或寻径问题（route planning/pathfinding problem）。今天，许多图应用程序充分利用图及其算法的非凡能力，而不再是通过尝试或错误排查来计算出网络上两个节点之间的最优路径。图 1-9 为一个简单的路径规划示例。

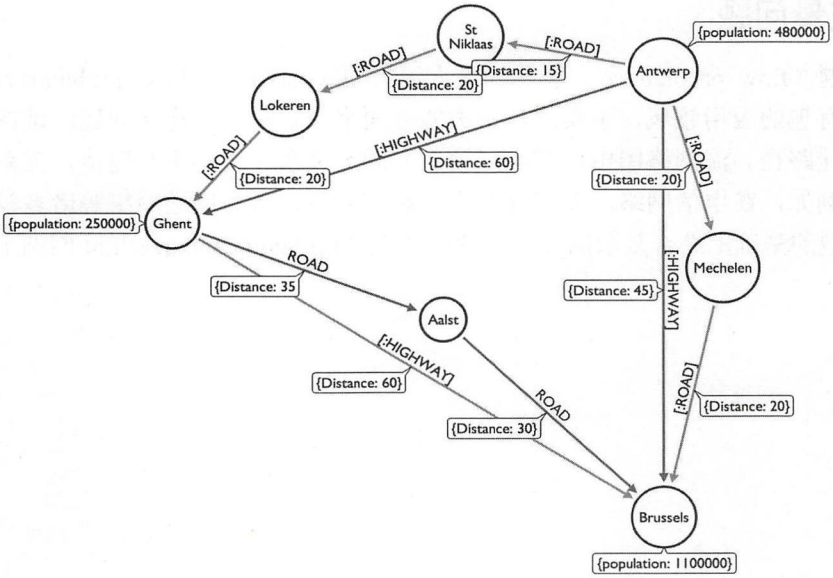


图 1-9　一个简单的含有普通道路和高速公路的城市路径规划示例

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

考虑一下物流领域一个非常简单的例子。假定将包裹从一个城市运送到另一个城市，需要关注以下几点：

- 城市间所有可用的路线。
- 最优可用路线，取决于网络中的各种参数，如容量、距离、成本、二氧化碳排放量、速度等。

这是一个很好的图算法用例。需要强调几个非常著名的算法：

- **Dijkstra 算法**：这是一个最有名的算法，可用于计算图中两个节点之间的最短加权路径，图中边上可赋予权重或成本。
- **A*（A-star）算法**：Dijkstra 算法的一个变种，使用启发式来更有效地预测最短路径搜索。当 A*算法发现潜在的图路径时，将其保存到一个带排序的优先队列中作为备选路径段，并计算出路径搜索过程中不同参数条件下**过去路径**（past path）和**将来路径**（future path）可能的开销。

根据所需的结果、特定的数据集和速度要求，不同的算法将获得不同的效果。

1.2.6　网页搜索

任何一本涉及图或图论的图书，哪怕是最优秀的，都无可避免地要提到世上最强大、使用最广泛的图算法——PageRank。PageRank 最初是一个图算法，是 1996 年由谷歌创始人拉里·佩奇在斯坦福大学发明的，用以获得更好的网页搜索结果。对于那些还记得互联网早期的搜索工具（例如 Lycos、AltaVista 等）的人们而言，它给终端用户带来了一个真正革命性的网页搜索方式。

图 1-10 为 PageRank 图示例。

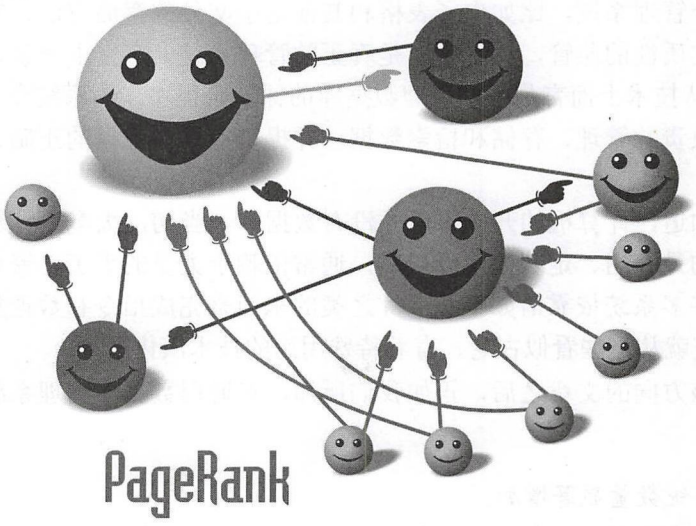


图 1-10　PageRank

原有搜索工具是在网页上进行关键词匹配，但谷歌彻底改变了这一做法，不仅仅关注关

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

键词，而且对不同网页之间的超链接进行连接分析。PageRank 和谷歌现今使用的许多其他算法都假定网页更重要，它们比搜索结果展现更丰富，因为大多数网页有大量来自其他网页的输入链接，从而可以通过分析网页之间的连接图来给页面打分。事实已经表明了 PageRank 算法的重要性。谷歌公司不仅利用此优势超越了雅虎一举成为最流行的搜索引擎，还在此图论算法的基础上建立了一个庞大帝国，其原理也应用到其他领域，如癌症研究、化学反应等。

至此，我们学习了图数据库的背景及其相关概念，下面需要更深入地认识以往的数据库管理系统和现今的 Neo4j 图数据库之间在渊源和操作上有哪些差异。

要做到这一点有必要介绍以下内容：

- 传统数据库的背景。
- 管理和存储数据的不同方法，包括老式的导航数据库、NoSQL、图数据库等。
- 图数据库的分类及其优缺点。

这将为后续学习奠定基础。

1.3 背景

现今难以查证，第一个真正意义上的数据库管理系统是何时产生的。自从巴贝奇（Charles Babbage）发明第一个完整的图灵计算系统（分析机，巴贝奇并未完成），我们一直认为计算机需要有某种记忆功能，用于负责处理所需操作和计算的数据。但是，什么时候这个内存演变成了一个合适的数据库呢？我们说的数据库到底是什么呢？

让我们先关注后一个问题。数据库可以被定义为任何有组织的数据集合。并不是所有的数据库都需要一个管理系统，比如电子表格和其他基于文件的存储方法，它们实际上并没有对数据采取任何实质性的监管，更说不上是真正的管理系统。广义上来说，数据库管理系统（简称 DBMS）从技术上而言是一组管理数据库的计算机程序。该系统介于用户可见的软硬件与数据之间，负责并管理、存储和检索数据，并提供对这些数据的正确、可靠、高效、安全的访问。

然而，我们知道，计算机的开始阶段并没有数据库。当初，大多数计算机都使用**内存**，而这种内存使用的是专用、定制的存储格式，通常依赖于大量的手工、繁重的劳动和基于硬件的存储操作。许多系统依靠诸如穿孔卡片之类的东西来完成指令和数据集的存储和处理。后来，计算机系统就从这些看似古老、有着特殊用途的技术演化而来。

在大量阅览该方向的文章之后，正如我们所知，对通用数据库管理系统的需求随着以下原因开始增长：

- 计算机系统数量显著增加。
- 计算机内存实现了一些突破。在 20 世纪 60 年代中期出现了直接访问存储器（Direct Access memory，一种无须依赖大量磁带或穿孔卡片的存储器）。

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

上述两个要素是任何一种多用途数据库管理系统存在的必要先决条件。第一个真正意义上的数据库管理系统出现在 20 世纪 60 年代，其快速发展的不同阶段都有着非常重要的意义。

数据库管理系统在半个世纪以来的发展可分为以下三个主要的阶段：

- 导航数据库（Navigational databases）
- 关系数据库
- NoSQL 数据库

下面我们将分别介绍图数据库的一些知识，以便更准确地认知图数据库，特别是 Neo4j，这才是本书真正的主题。

1.3.1　导航数据库

最初的数据库管理系统是由传奇计算机科学家查尔斯·巴赫曼（Charles Bachman）开发的，他对在极其稀缺的计算资源环境里如何构建软件系统进行了大量思考。巴赫曼发明了一种非常自然的方法（下面会看到图形化展示）来建模数据：采用网络进行关联。该数据库设计的出发点就是通常所说的**巴赫曼图**（参见图 1-11），其数据结构模型使人立刻想到与图形式（graph-like fashion）相似。

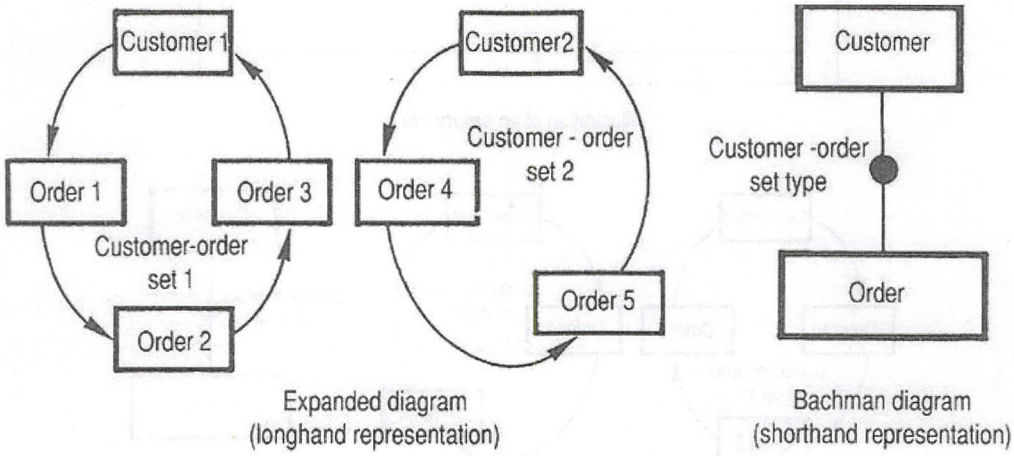
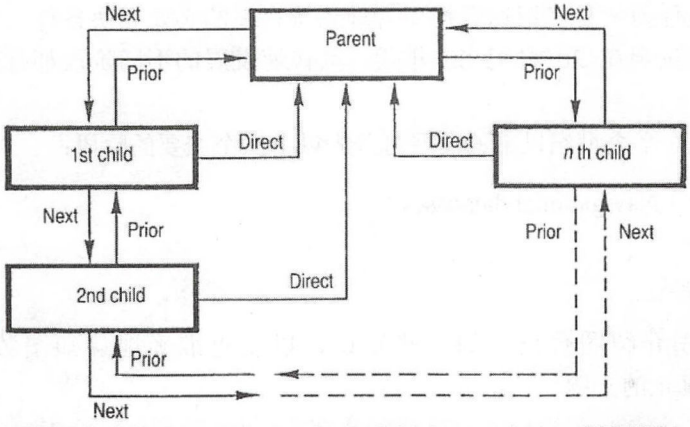


图 1-11　巴赫曼示意图

这些图表（diagrams）是数据库管理系统的起点，系统使用网状或层次结构作为其数据的基本结构。网状数据库和层次数据库系统的数据元素通过指针相互链接，如图 1-12 所示。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！



A closed chain of records in a navigational database model (e.g. CODASYL), with ***next pointers***, ***prior pointers*** and ***direct pointers*** provided by keys in the various records.

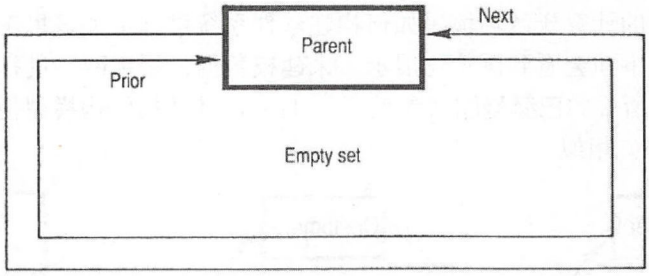


Illustration of an ***empty set***

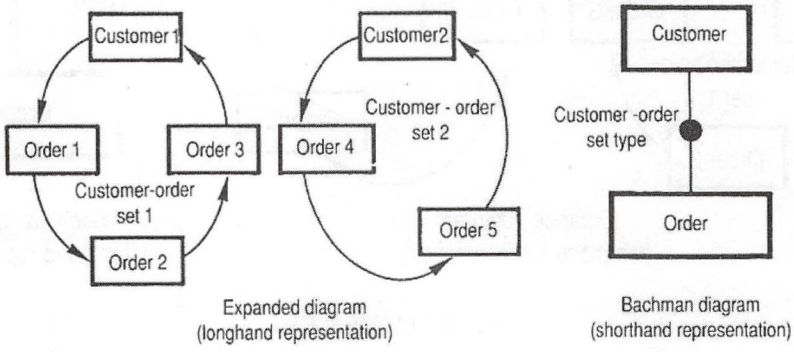


Illustration of a set type using a ***Bachman diagram***

The record set, basic structure of navigational (e.g. CODASYL) databse model. A set consists of one parent record (also called "the owner"), and n child records (also called members records)

图 1-12　一个带指针链接记录的导航数据库模型示例

如上所述，这些模型非常有趣，并给数据库行业带来了诸多影响。其中一项努力是**数据系统语言会议**（Conference on Data Systems Languages，CODASYL），它在 20 世纪 60~70 年代的信息技术行业中扮演了非常重要的角色，推出了世界上最重要的计算机编程系统——

非卖品，仅供非商业用途或交流学习使用
严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

COBOL，还为一系列导航数据库奠定了基础，如 IDMS、Cullinet 和 IMS。IBM 公司推出的 IMS 数据库是 CODASYL 网状模型的子集，通常归类为层次数据库。

导航数据库最终让位给了新一代数据库，即关系数据库管理系统（Relational Database Management Systems）。这一转变有许多原因，既有技术层面的，也有商业层面的，但似乎在整个行业内比较认同的两个主要原因是：

- 导航数据库模型的复杂性高。CODASYL 被广泛认为是只有绝对专家才能使用或理解——正如 1999 年经历的“千年虫问题”，需要许多类 CODASYL 专家加班加点，才能对系统进行移植以适应千禧年。
- 导航数据库缺乏一个声明式的查询机制。声明式查询机制本质上提供了一种非常重要的查找数据的方法：用户仅向数据库提供所需的查询，不需要告诉数据库怎么做，数据库自动提供答案。

上述原因致使导航数据库到关系数据库的过渡。

1.3.2　关系数据库

关系数据库管理系统可能是 21 世纪计算机科学中最熟悉的名词。该类数据库发明的背后历史很有趣。它由 IBM 圣何塞研究机构（IBM's San Jose，CA）一位不知名的研究人员埃德加·科德（Edgar Codd）提出。当时 Codd 在 IBM 硬盘研究项目中工作，导航数据库管理系统需要使用硬盘，后来才渐渐进入数据库领域。Codd 对这些系统越来越失望，主要原因是缺乏直观的查询接口。

事实上，可以将数据以网络或层次结构进行存储，但如何才能将数据提取出来呢？相关术语如图 1-13 所示。

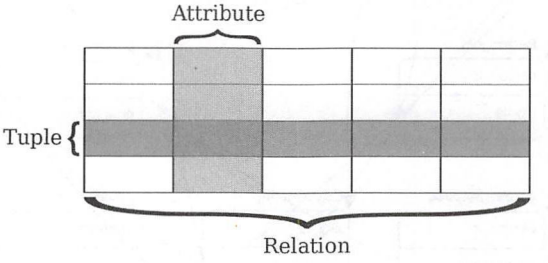


图 1-13　关系数据库术语

Codd 写了几篇有关数据库管理系统方面的文章，其方法不依赖于数据链接列表（网状或层次结构），而是更多地依赖于数据集。他提出了**元组演算**（tuple calculus）的数学表示方法，并证明与导航数据库管理系统相同。唯一的要求是，将产生一种合适的查询语言，以确保数据库的一致性要求。这成为了声明性查询语言（如**结构化查询语言**，Structured Query Language，SQL）的灵感来源。IBM 的 R 系统（System R）是第一个实现。后来前 IBM 员工、杰出的拉里·埃里森先生创办的小公司——软件开发实验室（Software Development Laboratories）在市场上最终击败了 IBM。这就是众所周知的，几年后关系软件公司（Relational Software, Inc.，即后来的甲骨文公司）发布的旗舰软件产品——Oracle。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

关系数据库提出了计算机科学研究人员所熟知的范式化（normalization）。该过程要求数据库建模人员通过依赖分析来减少数据冗余，节省磁盘存储，需将关系数据库表中多次出现的数据元素拆分到各自的表中。正如不会将 city（城市）作为 person（个人记录）的属性来存储，而是将 city 和 person 分为两个独立的表单独进行存储。这样，查询时将这些表进行关联，根据表之间的关系（1 对多，多对 1，多对多关系），引入一种新型的表（连接表，通常为两个表之间的多对多关系）来执行这些连接操作。

可以说关系数据库管理系统在过去的 30 年里为计算机行业提供了非常好的服务，而且很可能在未来很长一段时间内继续服务下去。然而，有必要指出，这也带来了一些问题，将再次为另一种数据库管理系统搭建舞台：

- 关系数据库系统受规模的影响严重。当关系集合或表变长时，关系数据库系统的查询响应时间通常会变得越来越糟。大多数情况下不是问题，但数据规模确实很重要，而且其不足的确会受到影响。举一个反面的例子，Google Spanner，它是一种可扩展、多版本、全分布、可同步复制的数据库。
- 关系数据库是非常“**反关系**”（anti-relational）的。比图数据库更缺乏关系。应用领域中采用关系模型建模会变得更加复杂、更难以处理。具体而言，join 操作在关系数据库管理系统用于从多个不同的集合或表中进行查询以获取数据，是极其复杂和耗资源的。系统能有效支持 join 操作的数量是受限的，在 **join 爆炸**（join bombs）到来之前系统已无法响应。请看一个 join 爆炸的例子（见图 1-14）。

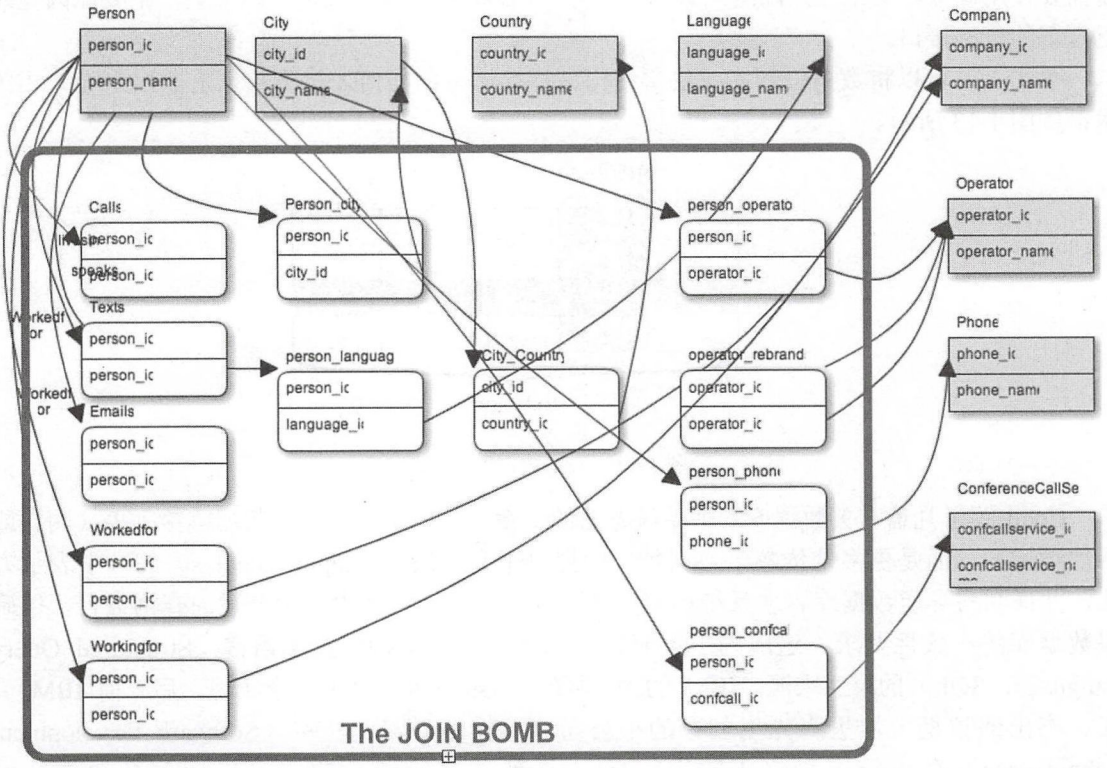


图 1-14　具有大量连接表的关系数据库模式

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

- 关系数据库甚至在数据入库前强加一个模式（schema），且该模式过于严格。每个用户都在域（domain）下进行操作，很难将单个数据库模式应用到整个域。从而需要一种更灵活的模式，以适应迭代式的敏捷软件开发需求。

接下来将看到，下一代数据库管理系统绝不会安于现状、止步不前，而会针对上述痛点不断创新。

1.3.3　NoSQL 数据库

“千年虫问题”和网页激增表明数据库管理系统新时代的来临。新一代数据库管理系统统称为 NoSQL 数据库，自然是针对关系数据库而言的。该名称的来源难以考证，但显然是关系数据库管理系统受挫后诞生的。大多数人将 NoSQL 当作 Not Only SQL 的缩写，但在数据库领域仍有争议。

我认为，大多数 NoSQL 数据库的基本理念是**面向任务**（task-oriented）的数据库管理系统。如同老生常谈：如果你唯一的工具是锤子，那么一切看起来都像钉子（If all you have is a hammer, everything looks like a nail.）。现在我们有不同种类的锤子、起子、凿子、铲子，还有更多的工具来解决数据管理问题。当然，最好的方法是选择合适的工具来完成不同的工作，如果只用关系数据库事实上可能会适得其反。除 SQL 数据库外，其他的可以分成四类：

- 键值存储（key-value stores）
- 列式存储（column-family stores）
- 文档存储（document stores）
- 图数据库（graph databases）

下面将分别介绍。

1. 键值存储

键值存储可能是最简单的面向任务的 NoSQL 数据库。其最初的数据模型并不复杂：主要基于亚马逊在两年一度的 ACM 操作系统研讨会（ACM Symposium on Operating Systems Principles）上发布的白皮书，一篇叫 Dynamo 的论文。在此讨论的数据模型就是亚马逊的购物车系统（Amazon's shopping cart system），该系统要求高可用和高负载。因此，键值存储数据库的底层数据模型的确很简单：键和值存储为无模式（schema-less）数据模型。事实上，该系统采用大量的商业硬件搭建成集群，可扩展性非常高，并承载了多个高端应用，比如 Amazon 等。键值存储的产品还有 DynamoDB、Riak、Project Voldemort、Redis、Aerospike 等。图 1-15 为键值存储的数据模型示例。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

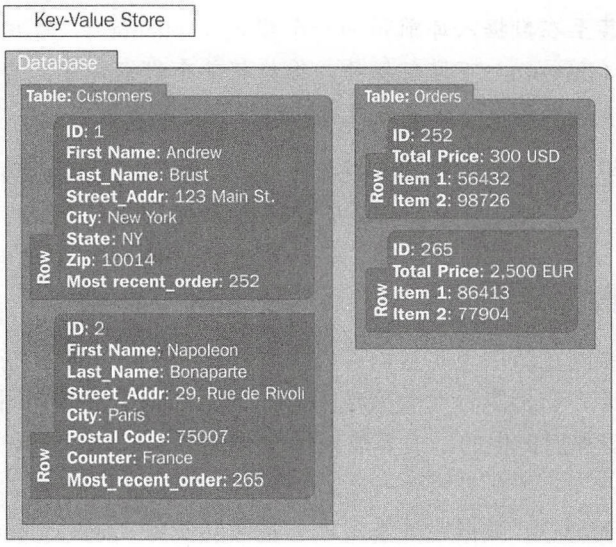


图 1-15　一个简单的键值数据库

2. 列式存储

列式存储是另一个面向任务的数据库解决方案。其数据模型比键值存储稍复杂，包含一个大而稀疏的表结构，其中包括存储键的多个列。与 Dynamo 系统类似，列式存储也是源于一个特定公司的特殊需求，即 Google 公司提出的解决方案，发表在 2006 年 OSDI 会议（Operating Systems Design and Implementation symposium，操作系统设计与实现研讨会）上的 BigTable 论文中。除谷歌的产品外，还涌现出一批有趣的开源实现，如 Apache Cassandra 和 HBase。大多数情况下，这些系统可结合 Map/Reduce 批处理来处理高级查询，如图 1-16 所示。

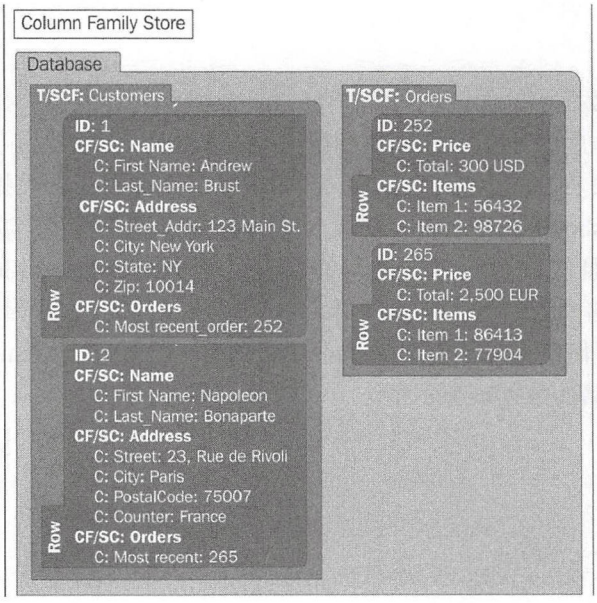


图 1-16　一个简单的列式数据模型

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

3. 文档存储

随着网页和应用的爆炸式增长，文档存储可能是最有名、最常用的 NoSQL 数据库类型。顾名思义，文档存储中的关键概念——**文档**，是一个半结构化的信息单元，可以是 XML、JSON、YAML、OpenOffice、MS Office，或者其他任何可用的文档。其存储和检索为简单的无模式方式。文档存储产品包括广受欢迎的 MongoDB、Apache CouchDB、MarkLogic 和 Virtuoso 等。下面给出一个简单的文档数据模型，见图 1-17。

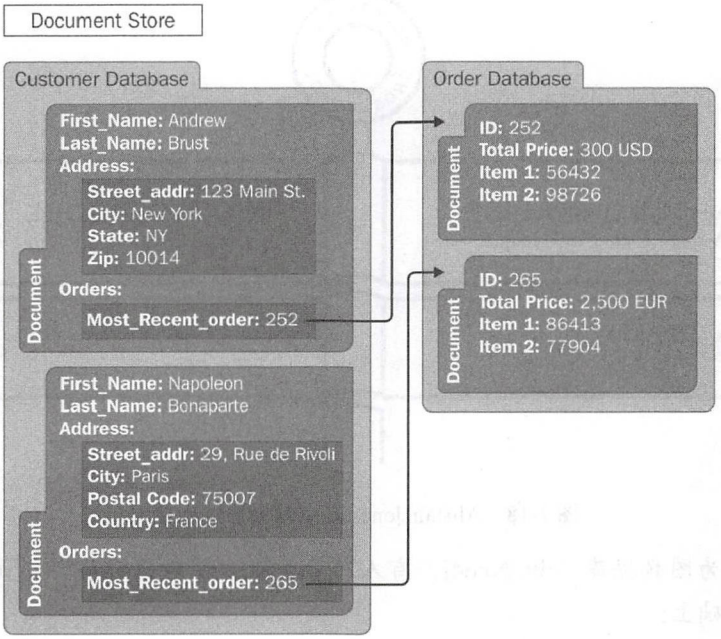


图 1-17　一个简单的文档数据模型

4. 图数据库

最后一类，也是本书的主题，即图数据库。通常也归为 NoSQL 数据库，但与其他三类有着本质上的差异。图数据库所要解决的问题与图和图论相关。图数据库，例如 Neo4j，其目的是为用户提供一种更好的方法用于管理结构复杂、呈网状分布的数据。

当然，基于图模型实现的解决方案并不仅有 Neo4j，有的产品成熟度不同，有的开源或闭源，例如 AllegroGraph、Dex、FlockDB、InfiniteGraph、OrientDB 和 Sones 等。

为更好地认识不同种类的 NoSQL 数据库，有必要按照各自的关键特性进行分类。为此，我借用了 Martin Fowler《NoSQL Distilled》书中和 Alistair Jones 演讲中所采用的基本分类方法。这两位学者和我的观点相同，NoSQL 数据库本质上依据对关系的不同看法可分为两类：

- 一类为聚合存储，包括键值存储、列式存储和面向文档的数据库。它们有许多共同点：
  - 都有一个基本的数据模型，基于一个单一的、丰富的、紧耦合的数据结构。在软件工程领域称为领域驱动设计，专家通常称作聚合（aggregate），因此这些 NoSQL 数据库都是面向聚合的数据库管理系统。

非卖品，仅供非商业用途或交流学习使用

- 都显然是对用例进行了优化，并在这些用例中保持读写一致，即“读到的即写入的”。比如想操作以前使用的数据——写入的键-值对、文档、行数据，这就需要应用级的处理，规模大的话还可能用到批处理。
- 都舍弃了关系数据库模型一个或多个特性，以获得其他优势。有的聚合存储放松了一致性或事务要求，以提高可扩展性。显然，如果更关注规模，那这种放松就不是什么事，如图 1-18 所示。

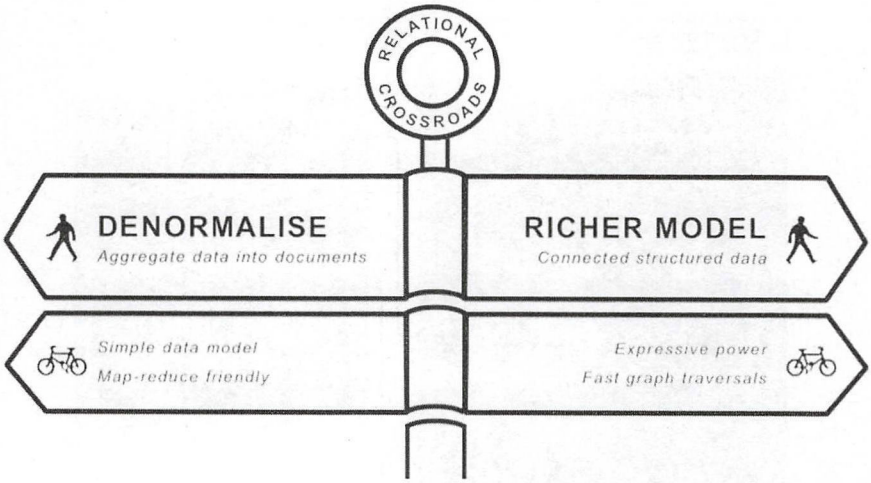


图 1-18　Alistair Jones 提供的关系转折点图

- 另一类为图数据库，如 Neo4j。有人可能会说，图数据库实际上是建立在关系数据库的基础上：
  - 数据模型中存储粒度更细、表现力更丰富，支持更复杂的查询，能有效解决“连接爆炸（join bomb）”问题。
  - 究其本意是重用了一些最初导航数据库的想法，并吸取了关系数据库的教训：降低复杂性、简化查询功能。

有了上述介绍和分类后，就可以更深入地研究图数据库了。

1.4 图数据库的属性图模型

如前所述，图数据库是独立的一类 NoSQL 数据库。更多的是因为其底层数据模型——属性图数据模型，有其独特的属性，需做进一步阐述。

首先，给出属性图数据模型的定义。本质上，图数据库也是用于存储数据。

图结构意味着将顶点和边（或者更倾向于称作节点和关系）进行持久化的数据存储。因此，图结构需完成以下操作：

- 以一种更自然的方式来表示数据，而不是对关系数据模型进行改造。

非卖品，仅供非商业用途或交流学习使用

- 在这些结构上构建各种图算法。

简而言之，图结构将数据具有图的性质视为其能力的重要组成部分，并将发现一个关键的功能，即图遍历：在节点和关系上进行遍历，并可通过连接节点的显式指针从一个节点跳到下一个节点。**免索引邻接**（index free adjacency），意味着无须进行索引查找就可以找到相邻的或相近的节点，也是性能提升的关键，将在后续章节中讨论。

重要的是，要认识到属性图模型并不适合所有的图结构。具体来说，它对以下内容进行了优化：

- 有向图：节点之间的连接（也称为关系）带有方向。
- 多关系图：两个相同节点之间可以有多个关系。稍后将介绍，这些关系明显不同，并且属于不同类型。
- 用键值对（key-value pairs）存储节点和关系的属性。

图结构元素可以有不同类型的属性，最基本的属性当然是顶点和边上的属性，见图 1-19。

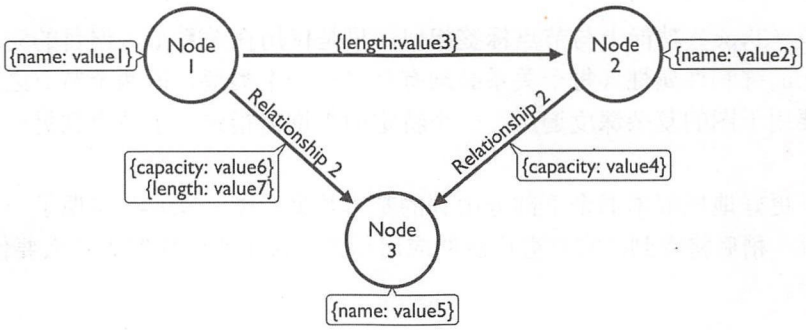


图 1-19　一个简单属性图示例

深入研究该模型将会发现以下几个方面很有趣：

- 没有固定的模式（schema）。数据库本身并不强制要求必须有一个模式，尽管大多数软件人员会赞同：想深入熟悉产品时，有某种模式并不是一件坏事。
- 部分归功于无模式特性，更适合处理半结构化数据。节点或关系可存储多个属性，无须单独设计，属性图就能自动准确地处理。
- 节点和节点属性易于理解。很容易地将节点对应到关系数据库术语中的表记录。属性图包含大量只含一条记录的表，也就是图的节点。节点属性，就如同表中的记录或行中的字段或列。
- 关系有点不同。关系总是有一个起点和终点，因此有方向，且不能为空，但可以是自引用（起点和终点关联的节点相同）。真正的优势如下：
  - **关系是显式的**：关系不是隐含在某种约束中，或通过连接操作在查询时构建。关系在图数据库中的地位与节点平等，有相同的表现力。
  - **关系可以有属性**：关系相关联的数值可以指定为该关系的长度、容量或任何其他特征。这点非常重要，与在关系数据库中所认知的完全不同。

非卖品，仅供非商业用途或交流学习使用

Neo4j 数据模型在属性图模型的基础上扩展几个关键概念。其中与属性图相关但不同的两个是节点标签（node labels）和关系类型（relationship types）。

1.4.1 节点标签

节点标签（Label）是对图中节点进行语义分类的一种方法。一个节点可以拥有零个、一个或多个标签，其使用方式类似于 Gmail 收件箱中的标签。在图结构中，标签本质上是一个集合概念：允许在数据库中轻松有效地创建子图，这有很多用途，比如仅查询部分数据库内容。最重要的是，可以使用标签在数据库中直接创建某种类型结构或模式，而不用自己去实现（在 Neo4j 2.0 之前版本需要自行实现）。带有一个标签的节点可对应于关系数据库表中的一行，而有多个标签的节点就没有与之对应的。虽然没有强制，但节点应该至少有一个标签。

1.4.2 关系类型

关系类型（Type）功能上与节点标签相似，只是仅用在关系上，但目的完全不同。关系类型是关系上的强制性属性（每个关系必须有且仅有一个类型，但两个节点之间可以有多个关系），主要用于图的复杂深度遍历，一个特定的查询需指定一个节点到另一个节点的特定路径类型。

这有助于更好地理解本书余下部分用到的基本数据模型。Neo4j 实现了一个非常完善的属性图数据库，稍后将看到，它有着广泛的应用场景。接下来将探究为什么要使用 Neo4j 之类的图数据库。

1.5 使用图数据库的注意事项

至此，应该对图数据库是什么以及与其他数据库管理系统和模型的关系有了很好的了解。本书大部分内容将详细介绍 Neo4j，它是图数据库管理系统的一个实现。在此之前，有必要与软件专业人员（开发人员、架构师、项目产品经理以及 IT 主管等）探讨一下为什么要使用图数据库。

事实上，在一些场景下，关系数据库与图数据库均适用，但还有许多其他类数据问题，关系数据库并不是专门为此设计的。接下来探讨哪些数据特性和查询模式适合图数据库。

1.5.1 为什么使用图数据库

当试图解决下列问题时，可以考虑使用 Neo4j 这样的图数据库。

1. 复杂查询

复杂查询本质上包含大量的复杂连接操作（join operations）。数据库管理员都知道，这

些连接操作在关系数据库中是非常耗时的，需要做笛卡儿乘积运算，两三个表之间的一两个连接操作还能正常运行，随表的数量呈指数增长，即便是小数据集也可能构成一个无法解决的问题，这就是为什么复杂查询成为问题的原因。

举一个复杂查询的例子，在一个伦敦社区里找出周日开放、为孩子们服务、提供印度餐的所有餐馆。从关系数据库的角度，这意味着要关联**餐馆**表（restaurant table）、**食物类型**表（food type table）、**开放时间**表（opening hours table）、**服务对象类型**表（caters table）和伦敦社区**邮政编码**表（zip-code table）后，才能给出查询结果。毫无疑问，还有很多其他的例子需要做类似的复杂查询，这只是一个假设。

在图数据库，连接操作将不复存在：需要做的是在图数据库中通过索引查找到一个起始节点（例如，London，伦敦），然后使用免索引邻接特性，从一个节点（London，伦敦）跳到下一个关联的节点（Restaurant，餐厅），其操作为：Restaurant -[LOCATED_IN]→ London。事实上，查询路径的每一跳都相当于一个连接操作。因此，图数据库中节点之间的关系也可以看作是关系数据库中连接操作的显式存储方式。

通常将这类查询称为模式匹配查询。指定一个模式（如图 1-20 所示：蓝色与橙色连接，橙色与绿色连接，蓝色与绿色连接），并将该模式锁定在一个或多个起点上，就可以查找出与该模式相匹配的节点。如你所见，图数据库将是一个很好的工具，可以围绕锁定的节点快速确定是否有匹配的模式。不匹配的模式将被忽略，不与开始节点关联的也不予考虑。

这实际上是图数据库性能提升的关键特性：一旦设定了一个起始节点，数据库将只在该起始节点关联的附近进行搜索，不相连的将完全忽略。这点表明：查询的性能与数据集大小无关，因为在大多数图中，并不是全关联的。同样，稍后将看到的，性能更多地依赖于结果集的大小，这也是在构建持久化体系结构时要牢记的关键点。

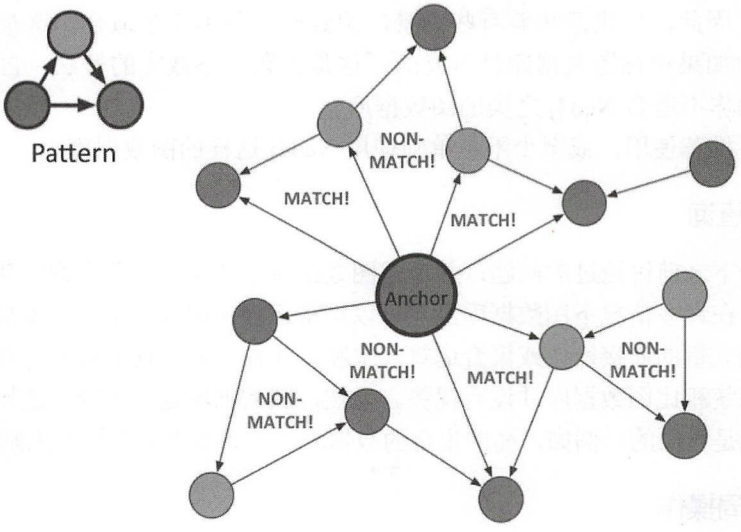


图 1-20　Anchor 节点匹配的模式

2. 实时数据的点击流查询

众所周知，如前面的示例，可以在不同的数据库管理系统中实现不同的数据查询。然

而，大多数查询在实时数据系统中的性能很糟糕，很可能影响整个应用系统的响应能力。因此，关系数据库采用特定的数据结构，并进行数据的预处理和预格式化。

这意味着在业务智能系统中经常需要数据去重、数据非规范化，以及使用 ETL 等技术（**提取、转换和加载**），以便对手头的数据创建特定查询的表示（有时也称为数据仓库中的立方体，cubes）。显然，商业智能行业价值不止 10 亿美元，这些技术都很有价值，但最适合于处理静态、很少更新的数据。图数据库能快速处理更多种类的复杂查询，数据可以是重复的或近实时的更新。

3. 路径查询

另一种非常适合于图数据库的查询类型是，查找不同数据元素之间是如何相互关联的。换句话说，就是在图中不同节点之间查找路径。在其他数据库管理系统中，这类查询问题需要清楚地了解可能路径的结构，也就是说，必须告诉数据库如何从一个表跳到另一个表。而在图数据库中，通常不需要这样做，虽然还可以采用上述方法，只需告诉数据库采用何种图算法、起始节点和终止节点，系统就将自动完成查询。图数据库自行判定数据是否连接以及如何连接，并采用路径表达式返回结果。事实上，将这些事情交给图数据库去处理是非常有用的，而且常常会带来意想不到、有价值的结果。

显然，上面提到的仅是查询类别，必须结合前面讨论的研究领域才能真正获得益处。在后续章节将作详细介绍。

1.5.2 什么时候不用图数据库以及用什么替代

正如之前所讨论的，NoSQL 数据库分类的核心都是以任务为导向。不同的任务选用与之相适应的工具。因此，这也意味着有些场景，图数据库并不完全适合。这难以得到图数据库粉丝的认可，但如果声称图数据库是万能的，这是愚蠢、不诚实的做法，自然不可信。有必要介绍哪几类操作不适合 Neo4j 之类的图数据库。

下列操作不推荐使用，或至少不是单独使用 Neo4j 这样的图数据库。

1. 大集合查询

如果回想一下之前讨论过的问题，并考虑图数据库是如何在复杂查询中获得性能优势的，就能立即发现，在许多情况下图数据库虽然可以正常运行，但效率不高。如果只是有效获取大量数据，不需要大量的连接操作或聚合运算（求和、计数、取平均值等），那么在性能上与其他数据库管理系统相比图数据库并没有优势。显然，图数据库也能执行上述操作，但性能优势很小，甚至可能是负面的。例如，面向集合的数据库，关系数据库在性能上则更高。

2. 图的全局操作

如前所述，图论从宏观上对图的分析和理解做了很多有意义的工作。查找节点集群、发现节点间未知的关系模式、确定特定图的中心等都是非常有趣的问题，但是与图数据库所擅长的还是差异较大。上述问题是从全局上去研究图，称其为图的全局操作。虽然图数据库在回答局部问题时非常强大，但另一类图工具（通常称为图处理引擎或图计算引擎）更适合求

解图的全局问题。

为完成这类全局性任务，这些工具通常是专用的，甚至使用特定的软硬件设备（常采用大内存、高性能 CPU），这也是另一种不同的 IT 架构。图处理通常是在后台分批进行，时间可能是几个小时、几天甚至几周，一般不用于基于 Web 的应用。这是一种完全不同的图操作类型。

3. 简单聚合查询

我们提到，图及图数据库管理系统非常适合复杂查询，而关系数据库不擅长。因此，在简单查询中，读写聚合数据通常用图处理非常低效，更适合采用面向聚合操作的键值存储或文档存储。如果复杂性较低，则使用图数据库系统的优势也会较低。

希望这些能让你更好地了解图数据库的优缺点。

1.6 问答

问题 1：图论是现代数学中一个很新的研究领域，由伦纳德·欧拉在 20 世纪晚期创立。

A.对　　B.错

问题 2：选出图不适用的当今科学或应用领域。

A.路由问题　　B.社会学研究　　C.会计系统　　D.生物学研究

问题 3：图是一种个别现象，只能应用于非常有限的一类应用或研究领域。

A.对　　B.错

问题 4：哪一类数据库与图数据库最相似？

A.导航数据库　　B.关系数据库　　C.列式存储　　D.无，图数据库是单独一类

问题 5：图数据库的数据模型通常为专有图数据模型，包含节点、关系和专有元素。

A.对　　B.错

问题 6：图数据库很适合处理简单的、面向聚合的查询应用。

A.对　　B.错

1.7 小结

本章介绍一些基本概念。这是 Neo4j 图数据库的基础，介绍了图的发展历史，解释了令

人着迷的数学领域图论的一些原则，并提供了一些其他领域的应用案例，这些领域已经从博大而长达一个世纪的发展的图论中获益。结论是简单明了的：图无处不在。世界大多真实互联，并且紧密相连，正如图术语所说。当然，这也影响计算机如何处理、如何存储数据以及如何交互。

这一章为深入了解图数据库这一奇妙世界奠定了基础。从整体架构上来看，有必要了解一下 Neo4j 这样的图数据库是如何产生的、想要解决什么问题、擅长什么以及可能不适合处理什么。

至此，我们将开始着手使用世界领先的图数据库——Neo4j。

下一章我们将关注计算机科学的特定部分——数据库管理系统的图结构。

第 2 章

Neo4j基础入门

在本章中我们将接触到这本书的真正主题：学习世界领先的图数据库 Neo4j。在本章，我们会熟悉 Neo4j 数据库的管理系统，在后续的章节我们也会结合实际模型和用例使用这个管理系统。

本章将要讨论的主题如下：

- Neo4j 的关键概念和特点
- Neo4j 的典型案例
- Neo4j 的认证许可模式
- 安装 Neo4j
- Neo4j 在云平台中的应用
- Neo4j 沙盒模式
- Neo4j 在 Docker 容器中的应用

下面让我们从第一个主题开始。

2.1 Neo4j 的关键概念和特点

在深入了解 Neo4j 之前，我们先来看看 Neo4j 作为图数据库管理系统的一些关键特性。相信这将帮我们指明并掌握 Neo4j 的一些关键优势。

2.1.1 从头开始创建图

像其他某些开源项目和开源 NoSQL 数据库一样，Neo4j 是为了解决某些特定的问题而被开发出来的。在 21 世纪早期，Neo4j 的创始人为某个媒体公司开发一套可以方便地管理媒体资产的项目，由于传统的技术栈无法达到预期效果，于是他决定采取一种激进的新方法来解决此项目中的问题，Neo4j 应运而生。

在早期，Neo4j 并不是一个功能齐全的图数据库管理系统，它更像是一个简单的**图表系统**，大家可以使用它的核心代码来以更简单的方式处理一些相互连接的数据结构，并且 Neo4j 在早期是开源项目。它基于传统关系数据库系统 MySQL 等之上开发而成，并注重为开发人员创建一个图抽象。显然，这样简单的功能还不能满足需求。过了一段时间，这个开源项目决定放弃使用 MySQL 作为基础框架，而从头建立一个图存储数据库。因为是从头开始重构，整个基础设施，包括用来存储图数据的二进制文件布局等底层组件都被重新优化，以

更好地处理图数据。这在许多方面看来是很重要的，因为它使 Neo4j 与其他数据库管理系统彻底区别开来。

对于本书的学习，我们不需要了解这个二进制存储文件的细节，但我们可以知道它是一个本地文件并面向图的存储格式，这种文件可以对图存储工作进行有效的负载调整。这是区别于其他类型数据库的很大不同之处。

2.1.2 基于事务的 ACID 数据库

ACID 是数据库事务正确执行的四个基本要素的缩写，即原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）。Neo4j 引以为傲的是它是一个标准的符合 ACID 的数据库。为了进一步了解这一点，我们有必要回顾一下 ACID 的具体含义。基本上，ACID 是许多数据库管理系统都共同遵守的最古老的标准，具体如图 2-1 所示。

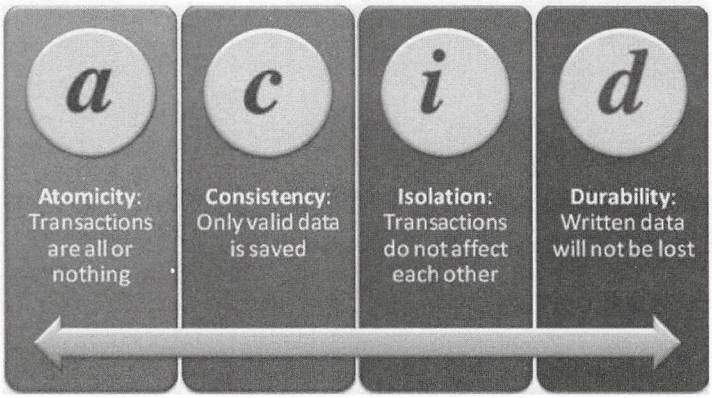


图 2-1 ACID 的含义

- **原子性（Atomicity）**：整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。
- **一致性（Consistency）**：数据库中的数据必须是一致的或**有效**的。在关系数据库中，如果事务是并发多个，系统也必须如同串行事务一样操作；在 Neo4j 中一致性实际上是图中关系必须有一个开始节点和一个结束节点，关系不能悬空，不能指向空或由空开始。在 Neo4j 中的一致性规则显得比较宽松，因为 Neo4j 实现了**可选**模式的概念。

在 Neo4j 中可选模式是一个非常有意思的机制。它是指在我们项目刚开始阶段，数据库中可以使用一个无模式的图结构，这在项目初期很有用。随着我们对项目涉及的领域认识不断深化，我们可以不断优化数据库中的图结构，因此我们无须在项目迭代初期就预定好图结构。然而，随着软件产品越来越成型，模式及数据一致性确实会很有用。在这点上，系统管理员和业务人员一样，在数据质量和一致性即 ACID 中的 C 方面进行权衡将变得更重要。Neo4j 完全支持上述两种方法，在当今敏捷开发方法领域中非常有用。

- **隔离性（Isolation）**：就是在同一个数据库实例上并行运行的多个事务不会相互影响，无论系统中发生什么情况，事务都需要互不影响地正常运行。隔离性特别体现在一个事务正在写入数据库而另一个事务正在读取数据的情况下。只要写入操作没有提交，读取操作将不得不使用旧的数据。
- **持久性（Durability）**：指的是已经被提交的事务不可丢失，必须持久地保存在数据库中。提交的数据和事务日志将被写入磁盘。现在大多数数据库包括 Neo4j 都能确保这一特性。

总而言之，Neo4j 真正被设计成为一个多用途的、功能完善的数据库解决方案。Neo4j 拥有目前传统数据库的大多数特性，并且它是使用图作为数据模型适应性存储、处理复杂关系的数据库。

2.1.3　可用于联机事务处理

联机事务处理是指可以在一个在线联机的环境中从 Neo4j 中请求并返回我们需要的数据。也就是说我们可以在一个网络请求中向 Neo4j 发送查询命令，Neo4j 会相应地在一个网络响应中返回我们查询的数据，并且这个请求与响应是以毫秒为单位的。

并不是所有数据库管理系统都需要这个特性。许多数据库管理系统实际上只需要响应一个请求，然后在几个小时后给出回复。在关系数据管理系统中，我们称之为分析系统。我们将这两种类型的系统分别叫作**联机事务处理**（OLTP）和**联机分析处理**（OLAP）。从概念和技术角度看两者之间存在一些差异。我们来比较一下，如表 2-1 所示。

表 2-1　OLTP 和 OLAP 的差异

	联机事务处理（OLTP）	联机分析处理（OLAP）
数据源	操作数据，OLTP 是数据的原始来源	合并数据，OLAP 数据来自各种 OLTP 数据库
数据操作的目的	可控制和执行基本业务	协助规划、解决问题和决策支持
数据可反映	正在进行中的业务流程的快照	多个业务活动的多维视图
插入和更新操作的性能	插入和更新由最终用户发起，且速度快、时间短	定期的长时间的运行批量作业来刷新数据
查询特性	查询较标准化且简单，返回数据量小	通常会涉及复杂的聚合查询
处理速度	非常快	依赖于所操作的数据量，批量数据和整库查询会很慢
空间需求	如果对历史数据进行存档则数据量很小	由于存在较大量的聚合数据结构和历史数据，因此需要比 OLTP 更多的索引
数据库设计	表较多且结构高度标准化	表较少且非标准化，常用星型和雪花型算法模型
数据备份与恢复	定期备份，数据对业务运营至关重要，如果数据丢失会带来重大的经济和法律责任	不需要定期备份，某些情况下可以简单地重新加载 OLTP 数据来恢复数据

在本书编写期间，Neo4j 明显属于 OLTP 类型的数据库系统。当然，这并不意味着你不能用 Neo4j 做任何分析任务，实际上，Neo4j 能更有效地运行一些分析任务（参见本章后续内容：Neo4j 典型案例）。由于 Neo4j 对处理分析任务没有做充分的优化，因此我们建议为了不影响正常的 OLTP 任务，那些分析任务最好运行在另一个单独的实例上。

2.1.4 扩展性能

为了能够承受高工作负载，Neo4j 显然也需要支持可扩展性、高可用性和容错性。通过创建数据库集群就可以达到这些目标，因此，Neo4j 企业版采用了集群解决方案，集群将在第 13 章“集群”中详细介绍。

2.1.5 声明式查询语言 Cypher

Neo4j 图数据库产品的一个重要特征就是它优秀的查询语言 Cypher。Cypher 是一种声明式的模式匹配查询语言，使图数据库管理变得易于理解、易于操作，即便是技术较差的用户也可以使用 Cypher 轻松操作图数据库。

Cypher 的关键特征就是它是一种**声明式**语言，与已经存在了相当一段时间的其他命令式查询语言不同。为什么说它这么重要？答案是：

- 声明式查询语言可以描述出我们在找什么，声明我们想要查看的模式，然后让数据库去处理如何去检索这些数据。
- 在命令式查询语言中，我们必须特别告诉数据库如何处理数据、如何检索数据。
- 声明式语言可以将解决问题的关注点分开，这样可以提高我们查询语句的可读性，这一点很重要，因为开发人员常常需要花很长时间去阅读数据库查询语句。因此，使用 Cypher 将使查询语句变得更易于阅读、易于与其他人共享、易于维护。
- 声明式语言能够按照数据的结构特性来优化查询从而能更有效地返回查询结果。也就是说，声明式语言可以进行查询优化，我们不需要考虑查询操作的内部细节。因此，声明式语言的查询速度更快。
- 声明式语言对于临时查询非常有用，我们无须创建中间存储（如关系数据库的临时表）。

我们之所以认为 Cypher 是 Neo4j 非常重要的一部分，是因为它是声明式语言，在数据库管理系统领域，特别是对于大规模的数据库应用来说是至关重要的。大部分开发人员并不想过多地考虑与数据库交互的细节，他们更希望专注于业务逻辑，也更希望数据库能简单、容易地提供所需要的数据。这也是关系数据库系统在 20 世纪 70 年代的进化目标（参见第 1 章），在图数据库管理系统领域也是以此作为进化目标的。Cypher 使我们可以更容易地操作图数据库，时至今日 Cypher 已经是一个令人难以置信的工具，随着时间的推移和 OpenCypher 的倡议，它会越来越好。Cypher 将在第 4 章“Cypher 入门”中介绍。

2.2 Neo4j 典型案例

与许多软件工程工具一样，Neo4j 也有其最佳的使用场合，充分利用它的优势可以为我们的项目添加很多有价值的特性。Neo4j 在两种特殊使用场合下，运用它的两种特定类型的数据库查询，能够有效解决问题，那就是复杂的连接（join）密集型查询和路径查询。

2.2.1 复杂的连接密集型查询

在前面的章节中，我们讨论了关系数据库管理系统在处理复杂连接关系的缺点，因为它们必须处理越来越复杂的数据模型。数据库引擎在查询中将会形成所涉及的表的笛卡儿乘积的操作，对于大型数据集，或者涉及两个以上的表格，此计算可能需要很长时间。

对于上述问题，图数据库管理系统是不会遇到的。节点与节点之间的关系是被有效预处理且显式持久化地存储起来的，数据之间的连接关系就像链表中从一个节点跳到另一个节点一样简单。在关系数据库中非常困难的关系连接问题在图数据库中变得非常简单、高效、快速。

2.2.2 路径查询

Neo4j 的许多用户经常用它的图数据结构来寻找不同节点间是否存在路径。我们在做这样的操作时通常会有以下问题需要解决。

- 路径是否真的存在：两个数据元素之间是否有连接？如果有，这个连接是什么样的？
- 寻找最佳路径：两件事物之间的哪条路径是最优路径？
- 路径查找的可变性：如果节点或关系的属性发生变化，路径会发生什么变化？

这两个典型案例都有一些重要的特征：

- 图数据都是**存储在本地**，并且在图数据库中开始遍历的图具有一个或多个固定的起始点（或**锚点**）。
- 是接近实时的对数据进行操作（与商业智能中称为多维数据集的复制数据操作不同）。

Neo4j 作为图数据库管理系统较为成功的另一个关键要素：它是一个开源解决方案。

2.2.3 开源解决方案

长期以来，在企业信息技术领域采用开源技术来实现许多关键业务是非常重要的事情。采用开源技术经历了至少十年的演变过程，从网络服务器等外围系统开始，到关键任务操作系统、内容管理系统、CRM 系统、数据库系统（如 Neo4j）。采用开源解决方案对企业应用是非常有利的决策。

使用开源框架是非常有利的，比如：

- **不会受框架供应商制约**：由于源代码是开放的，因此我们自己可以查看、修改源代码，这样我们可以更灵活地使用它。
- **更高的安全性**：由于代码可接受公众审查，原始开发人员就不可能在源代码中植入后门程序，因此开源软件系统本质上应该是更安全的。
- **更易于故障排除**：由于开源框架供应商和用户都可以访问源代码，他们之间可以交换系统的详细调试信息，因而如果出现问题那么查明问题会变得更容易。
- **利用可扩展性实现更多创新**：通过公开源代码，许多人可以使用此开源框架，并且在使用过程中开发人员会贡献出使用中遇到的问题的解决方案，这样此开源框架就会被充实。
- **具有研究价值**：开源框架（即使是已经具备商业价值的项目，如 Neo4j），通常允许研究人员免费使用该框架，并且研究人员可以对其进行再加工后再次开源。
- **更节省经济成本**：开源框架倾向于使用**公平的**许可模式，用户使用开源框架并从中获益后才需要支付一定的费用。这样在用户的项目初期就不必担心购买框架的钱入不敷出。

以上益处，Neo4j 都是具有的。

2.3 特性

Neo4j 作为图数据库管理系统，提供了两个不同版本：

- **社区版（Community Edition）**：这是基本的、功能齐全的高性能图形数据库。
- **企业版（Enterprise Edition）**：企业版相比社区版增加了许多典型的企业功能，如集群（高可用集群和负载均衡集群）、高级监控、高级缓存、在线备份等。

Neo4j 大多数用户都是从社区版开始开发应用，然后在生产环境下使用企业版。

从企业版本 3.2 以来，可以将 Neo4j 作为集群进行部署，集群之间可以在地理位置上无限制，并确保事务处理的 ACID 特性。

技术支持

Neo4j 不同的版本有不同的支持渠道：

- 社区版提供**社区支持**。你可以在公共论坛（Google Group、Stack Overflow、Twitter 和其他渠道）提问和寻求支持。但是请注意以下几点：任何用户的问答都是公开

的，用户的答复不能保证完全正确、实时，Neo Technology 赞助了一批顶尖工程师帮助用户解决问题。

- 企业版提供一个专业的技术团队支持。企业版用户可享受官方技术团队的专业支持，官方技术团队会按照优先级来为企业级用户提供服务。另外，企业用户还可以直接与 Neo4j 产品工程师沟通，以便用户获得第一手信息。

另外，Neo4j 也提供了一个帮助系统，在用户刚开始使用 Neo4j 的时候会有很多疑问，通过帮助程序可以获得相关知识信息。

2.4 许可协议

Neo 官方已经为 Neo4j 规定了非常具体的许可条款，这些许可协议的目标为：

- 推广开源软件。
- 促进 Neo4j 的社区发展。
- 通过提供收入来源确保 Neo4j 项目的长期开展。

许可协议是通过以下方式实现的：

- 社区版使用 **GNU 公共许可证（GPLv3）**作为其许可协议。这意味着只要你在 GPL 下进行修改，就可以复制、分发和修改 Neo4j 社区版本的代码。你可以使用 GPL 协议将 Neo4j 产品应用在商业上，但是你必须提供源代码。GPL 要求你公开源代码，并且前提是代码必须通过 Java API 直接与 Neo4j 代码进行交互。如果你使用的是 REST API，因为没有对源代码进行修改，所以可以随意使用 Neo4j。
- 企业版使用双许可证模式。这意味着 Neo4j 企业版的用户可以由以下两种选择：
  - 使用 Affero GNU **公共许可协议版本 3（AGPLv3）**，有时也被称为**网络版本的 GPL**。

AGPL 许可协议不同于其他 GNU 许可协议，因为它是专为网络软件而构建的。AGPL 除了具有 GPL 相同条款外，它不仅限制开发人员在同一台机器上（通过 Neo4j 的 Java API）使用 Neo4j 的代码，还限制通过网络与 Neo4j 进行交互（通过 Neo4j 的 REST API）。所以，这意味着如果我们免费使用 Neo4j 的企业版，则必须开放我们的源代码。

  - 获得 **Neo 技术商业许可证（NTCL）**。此许可证是典型的商业许可协议，它允许我们在特定数量的计算机上在特定时间段内使用 Neo4j 企业版。

Neo4j 官方提供了许多不同商业许可协议供我们选择，具体取决于你将要部署的实例的数量、你所在公司的类型（教育、中型企业或大型公司）、法律合同要求协议和支持合同。有关这些商业许可协议的更多信息，请联系 licensing@neotechnology.com。

了解了以上内容后，我们就可以开始讨论如何在不同平台上使用 Neo4j 了。

2.5 安装 Neo4j

在本节中，我们将介绍开始使用 Neo4j 所需的前几个步骤。这些步骤在不同的平台上有很大的不同。对大多数人来说，这将是一个简单的步骤，但也是我们不能跳过的重要步骤。

2.5.1 在 Windows 系统上安装 Neo4j

在 Windows 系统上安装 Neo4j 首先需要去官方网站下载安装包（http://www.neo4j.org/download），在官网下载页面找到最新版下载链接（见图 2-2）。

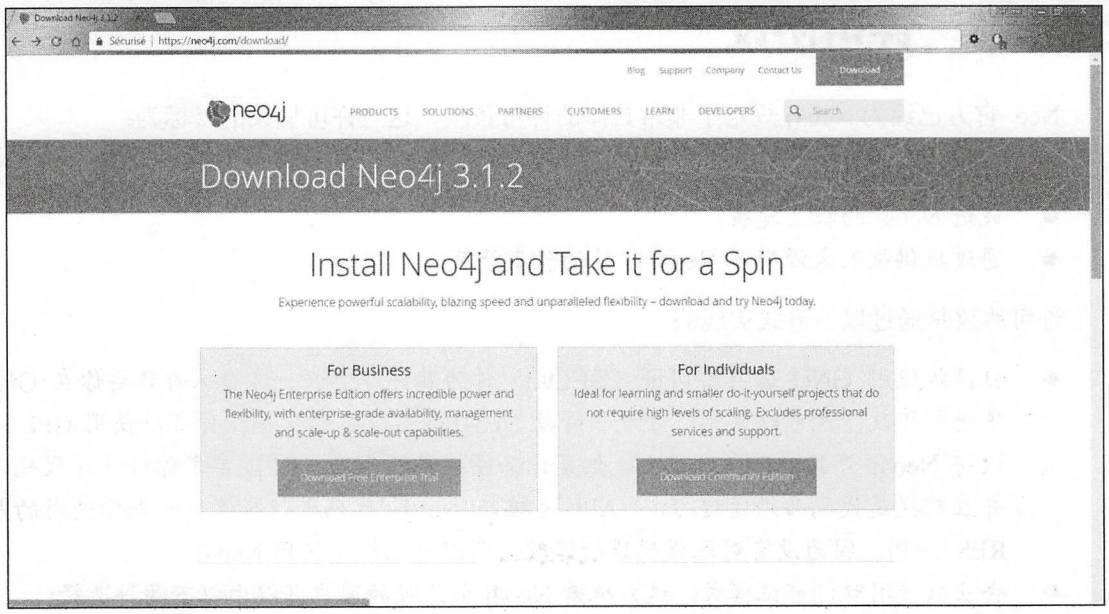


图 2-2 Neo4j 下载页面

目前，Neo4j 在基于 webkit 的浏览器上表现最好，比如 Chrome 浏览器（我们将使用 Chrome 讲解）。

Neo4j 社区版为我们提供了一个极好的入门平台，在 Windows 上 Neo4j 安装包为我们提供了一个简单、便捷的安装方式，安装包运行时如图 2-3 所示。

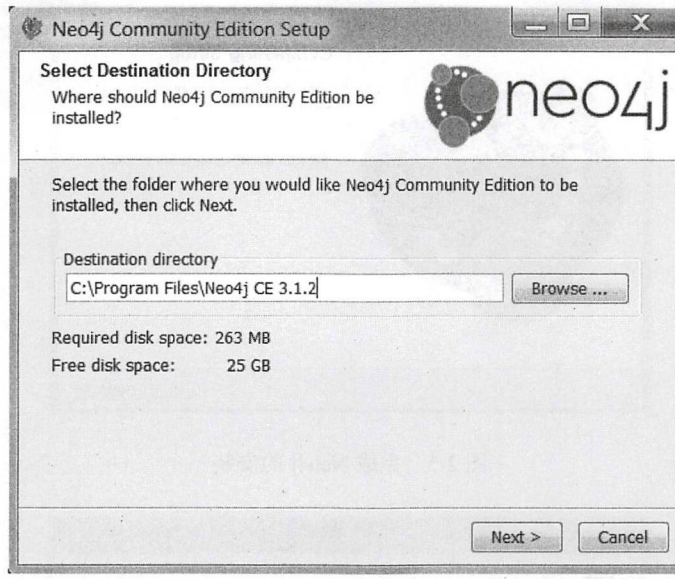


图 2-3　Neo4j 安装包

Neo4j 的 Windows 安装包为我们提供了安装 Neo4j 所必需的各种配置。在接受许可协议（见图 2-4）后，安装向导会很快运行 Neo4j。

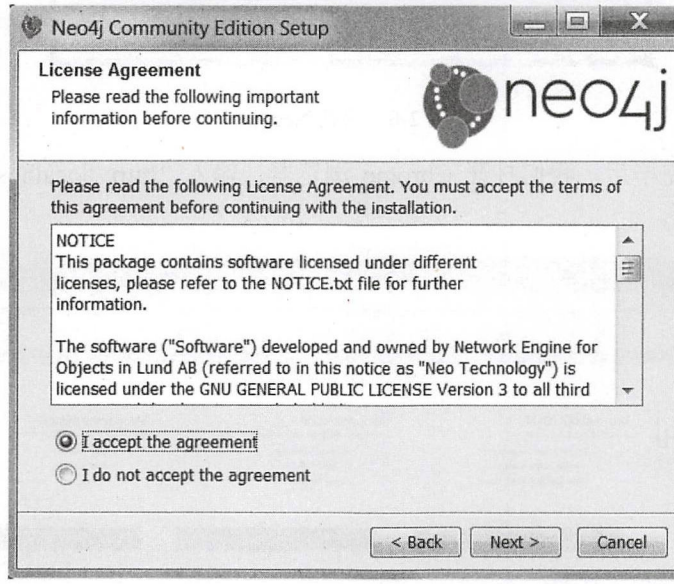


图 2-4　接受许可协议

下面我们会看到，安装程序完成安装（见图 2-5）后，Neo4j 会马上启动（见图 2-6）。

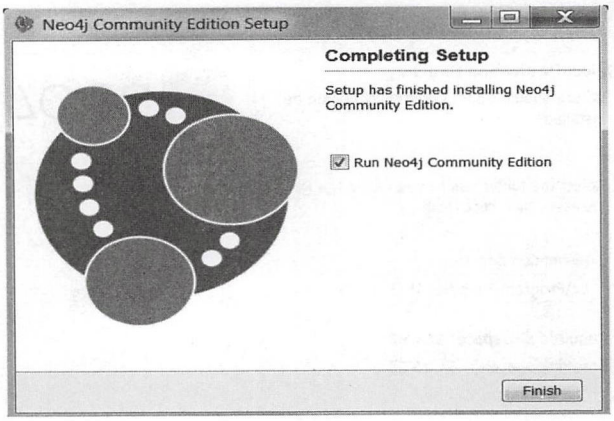


图 2-5　完成 Neo4j 的安装

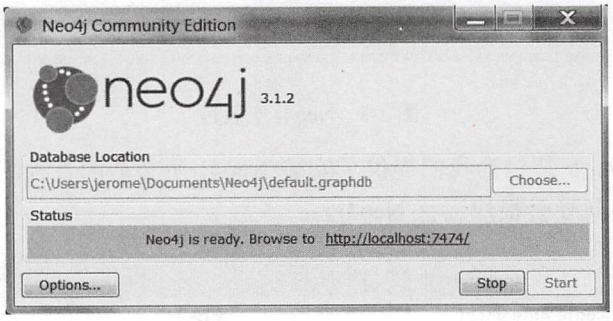


图 2-6　启动 Neo4j

当 Neo4j 启动完毕后，我们打开 Chrome 浏览器，输入“http://localhost:7474”，将打开如图 2-7 所示的界面。

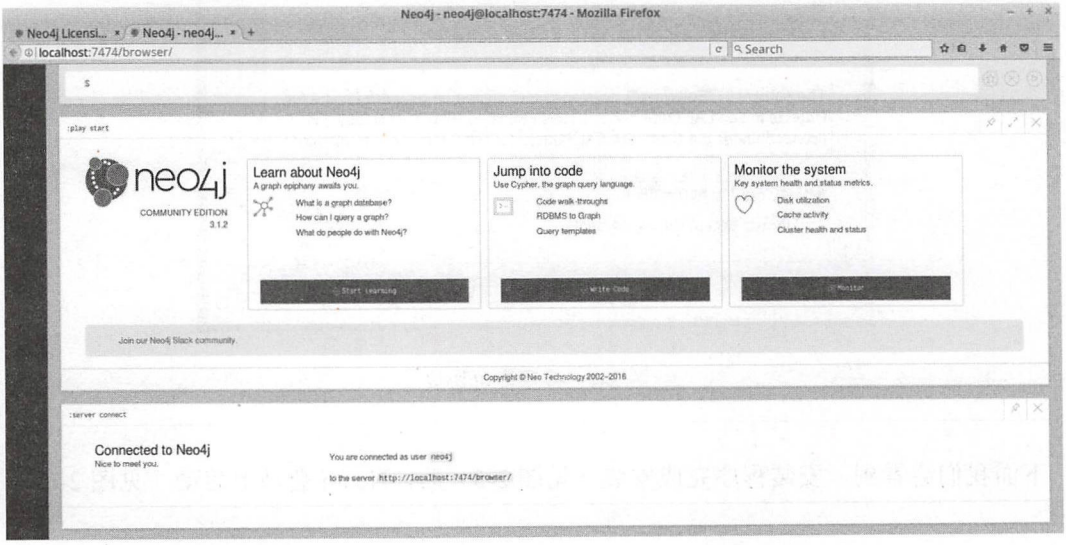


图 2-7　Neo4j 登录界面

Neo4j 安装完成后，如果想要寻找 Neo4j 的工具文件和执行文件，可以到 C：\ Program Files \ Neo4j 下找到。

了解以上知识后我们就可以在 Windows 系统上使用 Neo4j 了。

2.5.2　在 Mac 和 Linux 系统上安装 Neo4j

Mac 系统和 Linux 系统上的 Neo4j 安装包下载是相同的，打开 Neo4j 官网下载页面（见图 2-8）。

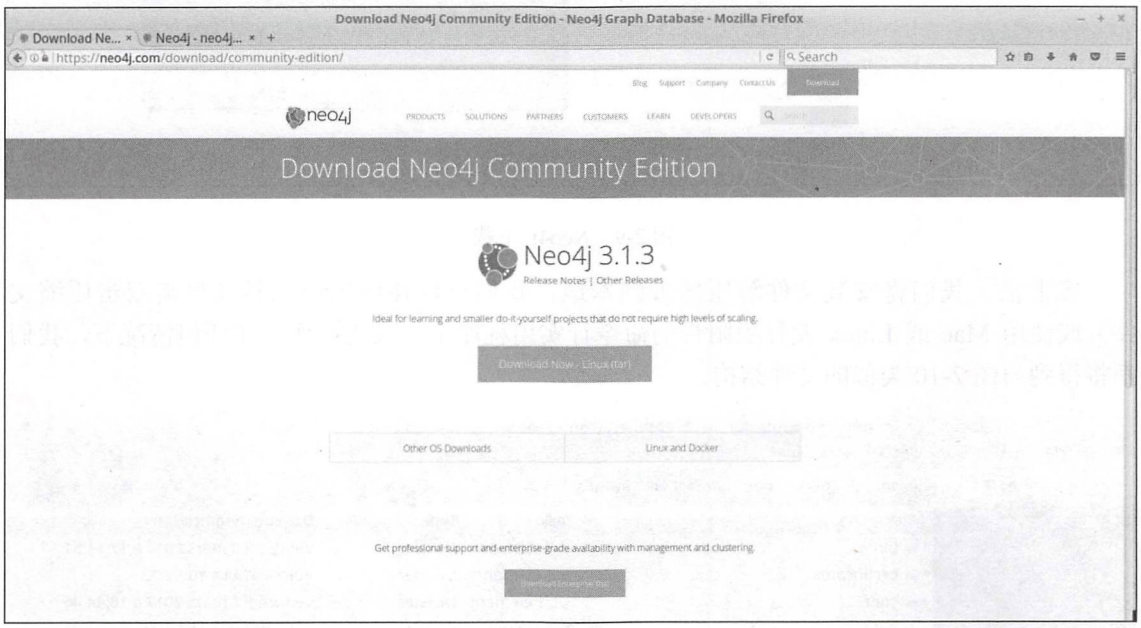


图 2-8　下载 Linux 和 Mac 版本 Neo4j

对于 Linux 和 Mac 版本的 Neo4j 安装，需要注意的是安装包并不包含 Java 虚拟机。在上一节中我们介绍了 Windows 上安装 Neo4j 是不需要额外安装 Java 虚拟机的。但是在 Mac 和 Linux 上，我们需要确保系统上已经装有 Java 7 及以上版本的 Java 虚拟机。所以我们需要事先安装 Java 虚拟机，具体步骤请到网上查阅 Java 虚拟机安装步骤。

在这里我们使用 Linux 作为实例来介绍，对于 Mac 系统安装步骤是相同的。

首先，从下载包含 Neo4j 的文件开始，如图 2-9 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

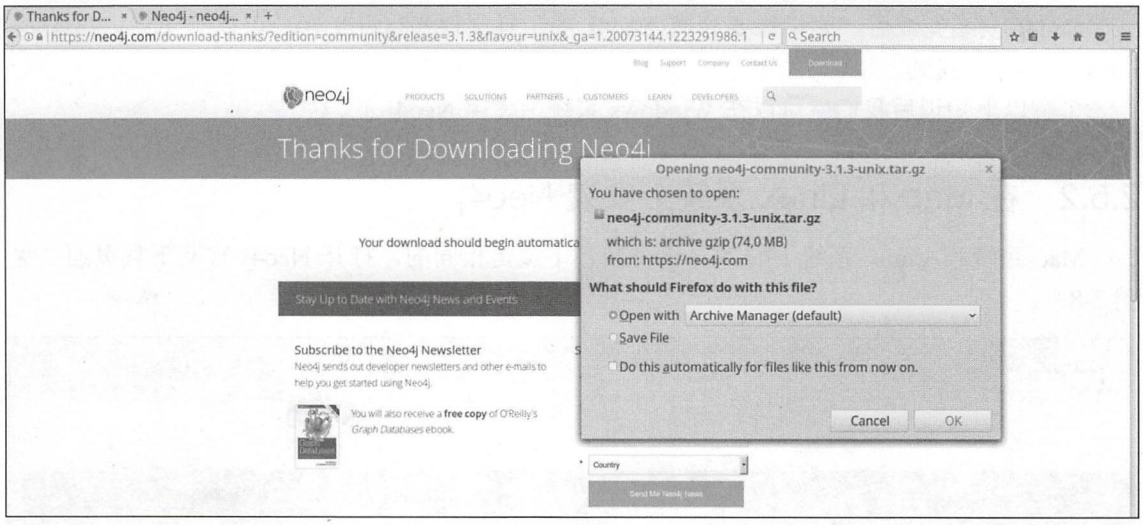


图 2-9　Neo4j 下载

接下来，我们将安装文件解压部署到本地。我们可以使用图形工具（只需双击压缩文件）或使用 Mac 或 Linux 发行版附带的命令行实用程序来完成此操作。在任何情况下，我们都将得到与图 2-10 类似的文件结构。

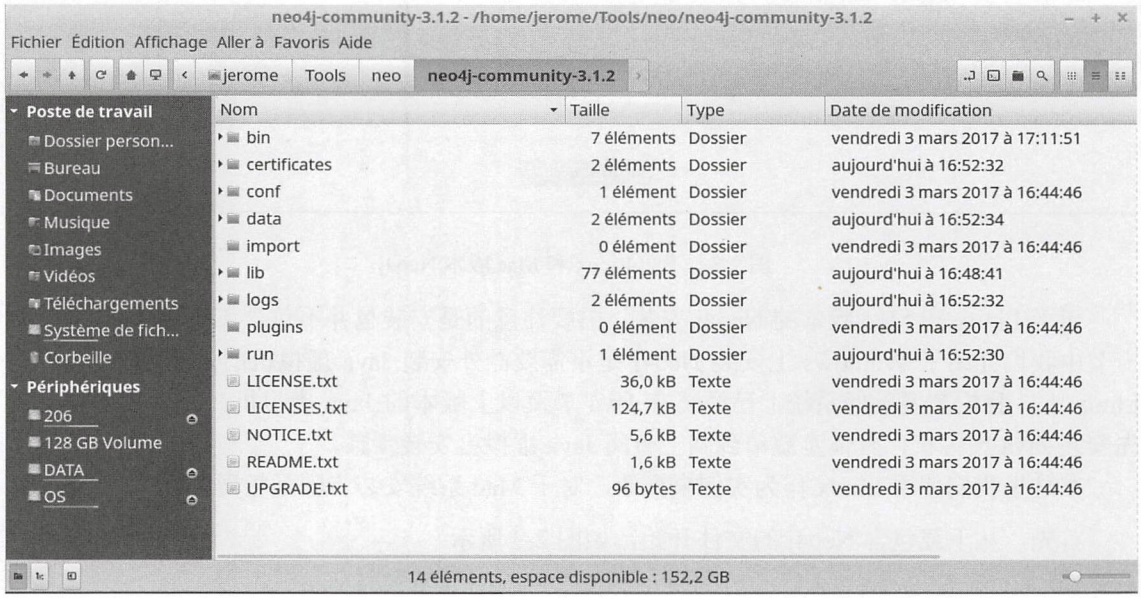


图 2-10　解压部署后的文件结构

接下来，我们打开命令行终端，导航到 Neo4j 的执行目录下，然后通过运行一个简单的命令“bin /neo4j start”来启动 Neo4j，如图 2-11 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
jerome@nitendet ~/Tools/neo/neo4j-community-3.1.2/bin
Fichier Édition Affichage Rechercher Terminal Aide
jerome@nitendet ~/Tools/neo/neo4j-community-3.1.2/bin $ ./neo4j start
Starting Neo4j.
WARNING: Max 1024 open files allowed, minimum of 40000 recommended. See the Neo4
j manual.
Started neo4j (pid 23658). It is available at http://localhost:7474/
There may be a short delay until the server is ready.
See /home/jerome/Tools/neo/neo4j-community-3.1.2/logs/neo4j.log for current stat
us.
jerome@nitendet ~/Tools/neo/neo4j-community-3.1.2/bin $
```

图 2-11　从命令行工具运行 Neo4j

与 Windows 版本的 Neo4j 安装相同，我们只需要用浏览器打开 http://localhost:7474 网址就可以看到 Neo4j 的操作界面了。

Neo4j 默认安装带有一个简单的数据库实例和一个教程，通过这个附带的实例，我们可以在本章接下来的内容中继续学习。

2.6 在云平台使用 Neo4j

在本节中，我们将讨论在云平台用 Neo4j 图数据库管理系统。我们可以在实际操作中体验到 Neo4j 数据库解决方案的强大功能，并且无须通过前面提到的在你自己的操作系统上安装 Neo4j，而是可以使用以下解决方案：

- GrapheneDB。
- Heroku。
- GraphStory。
- GraphGrid。
- Azure。
- 在 CleverCloud、OVH、Google 云平台、OpenShift、亚马逊网络服务（AWS）等平台安装 Neo4j。

我们可以自己选择一个 Docker 托管服务提供商，从而构建自己的图数据库，有关怎样使用 Docker 安装 Neo4j 的细节会在后续内容讨论。

云平台入门包含几个简单的步骤：

步骤 01　注册 GrapheneDB（见图 2-12）。这很容易，因为他们提供了一个免费的模板来对我们的项目进行使用和测试。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

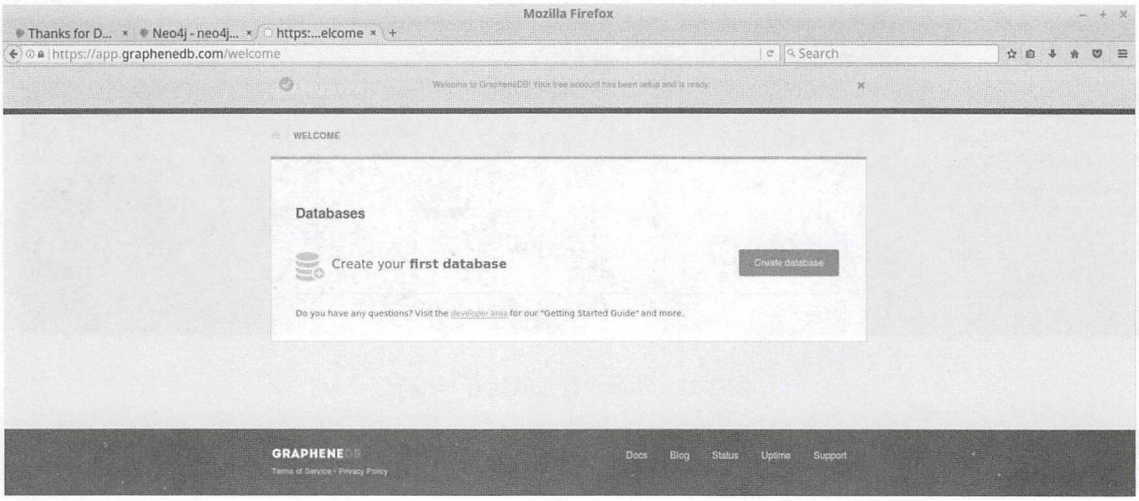


图 2-12　开始使用 GrapheneDB

步骤 02　一个数据库实例（见图 2-13、图 2-14），这相当于启动一个 Neo4j 服务器，但这并不在我们的计算机上。

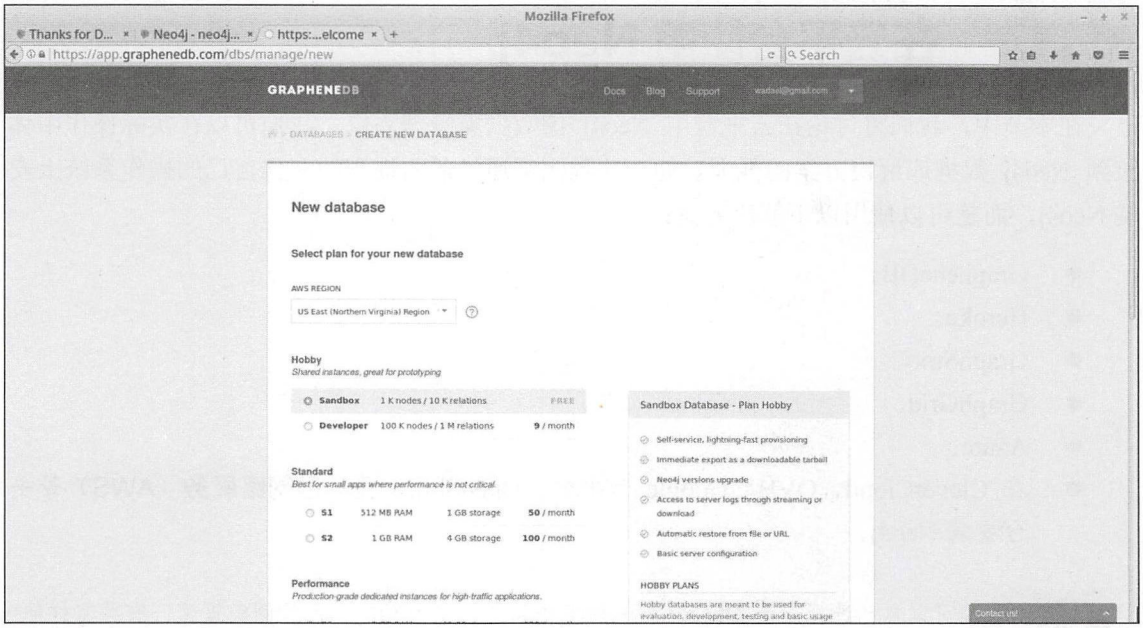


图 2-13　创建一个数据库实例 1

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

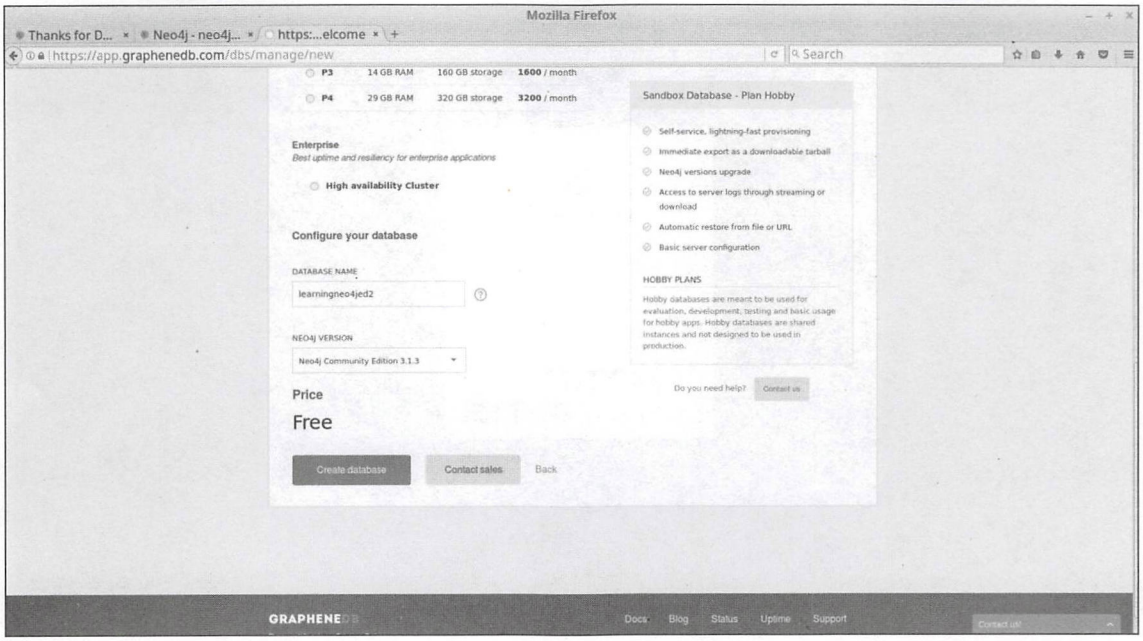


图 2-14　创建一个数据库实例 2

步骤 03　数据库实例创建完毕后，我们就可以创建一个用户，如图 2-15、图 2-16 所示。

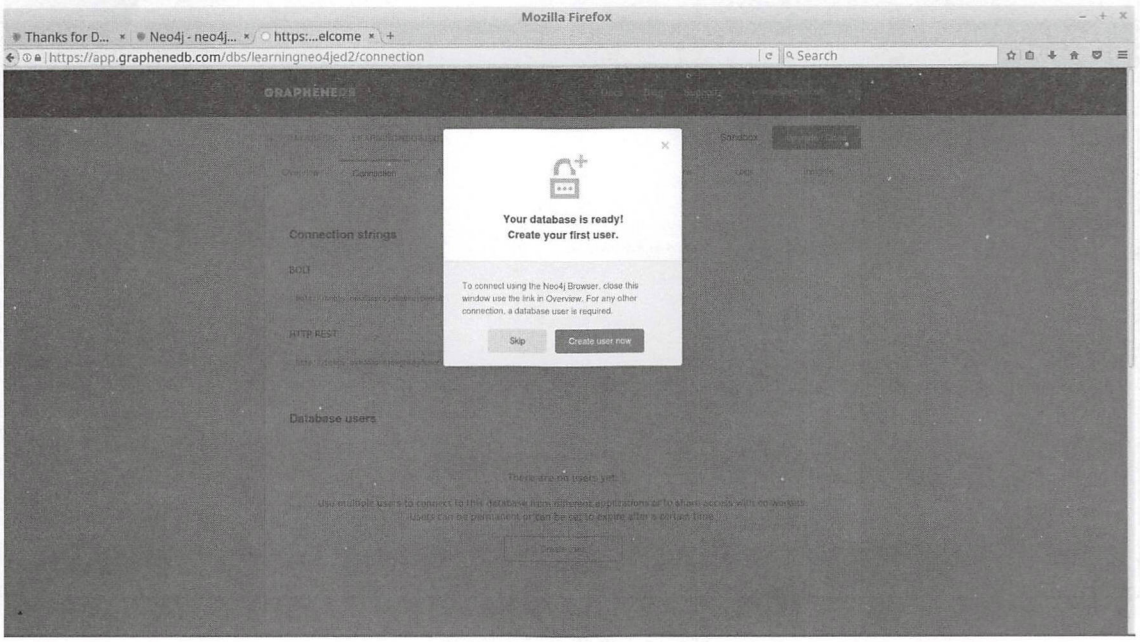


图 2-15　创建一个新的 Neo4j 用户 1

步骤 04　在有限的使用许可范围内我们创建的用户是不会过期的，图 2-17 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

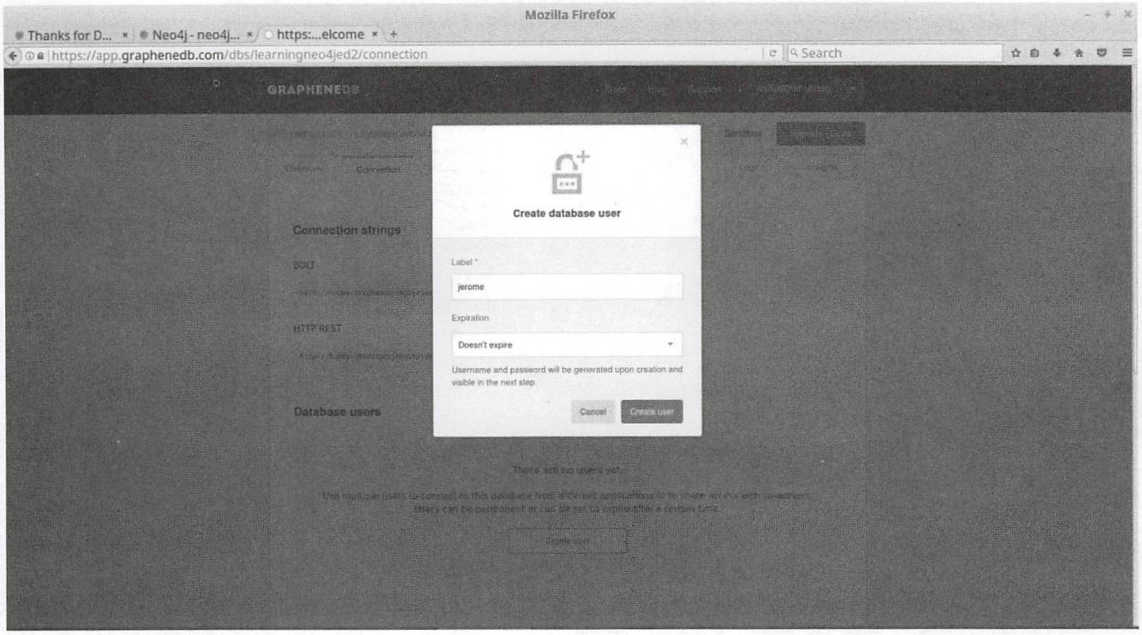


图 2-16　创建一个新的 Neo4j 用户 2

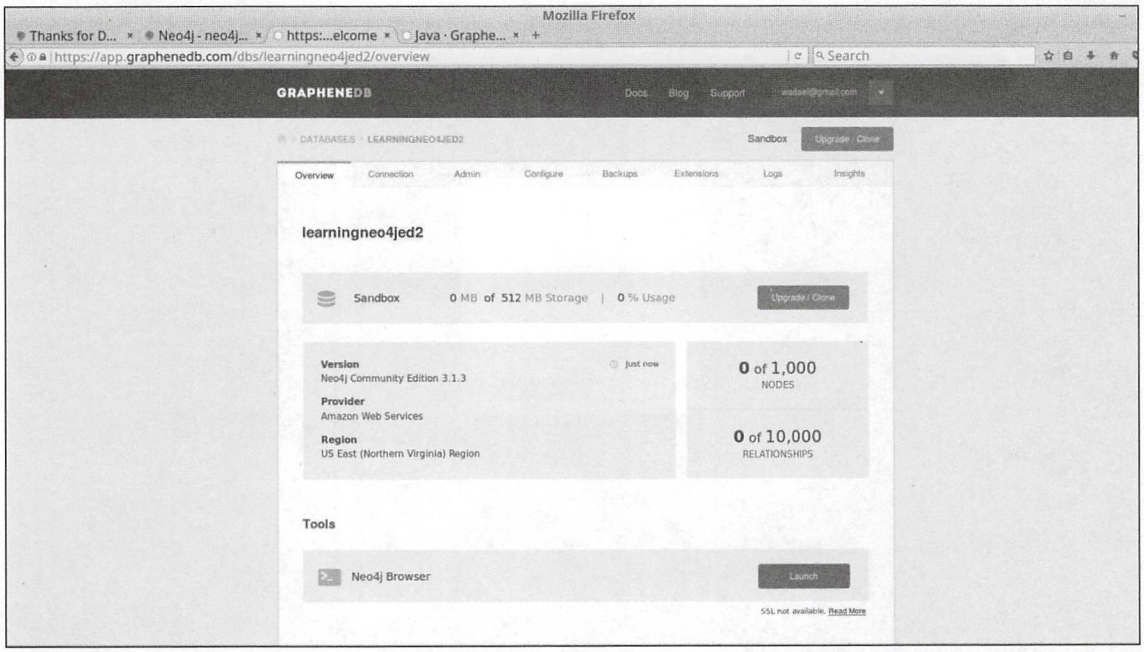


图 2-17　启动 Neo4j 数据库

步骤 05　验证后，进入“概览”选项卡。单击工具部分的启动按钮。

步骤 06　接下来我们用浏览器访问 Neo4j 的控制台界面（见图 2-18），在这个浏览器界面中，我们可以看到它有用户名、密码保护；登录后的界面让我们感到很友好、熟悉，这与我们在本地装的 Neo4j 的控制台界面是一样的。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

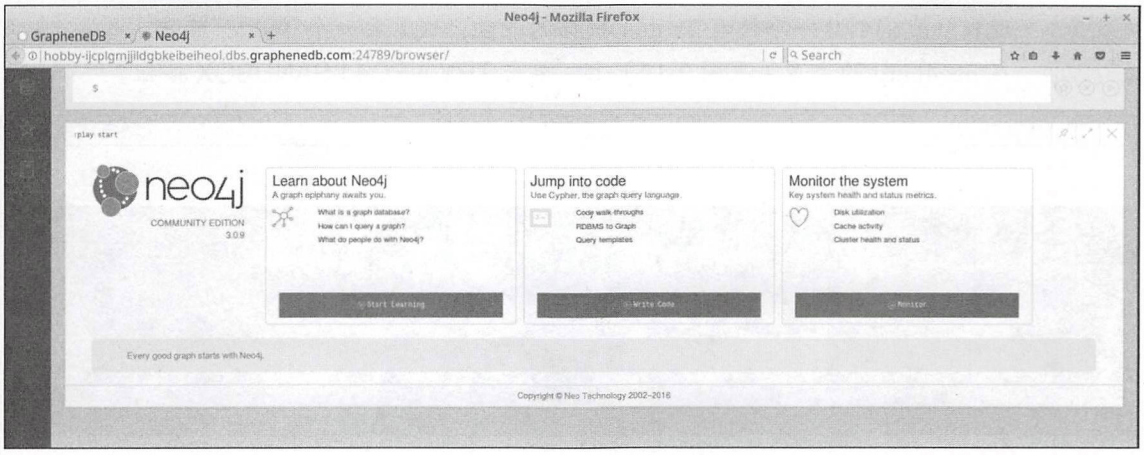


图 2-18　Neo4j 控制台界面

步骤 07　在云平台上使用 Neo4j 与在本地使用也是有点差异的，其中之一就是访问 REST 界面的方式。

步骤 08　最后，我们可以使用这个云平台的 Neo4j 系统来做许多有意义的事情，例如执行数据库的导入和导出操作。这是一个很重要的功能，它可以将我们在本地创建的数据库数据传送到 http://www.graphenedb.com/上的 Neo4j 实例上（在 Admin 标签下，查找**导出数据库**按钮）。

通过上面的学习，我们希望能了解在云平台上使用 Neo4j。如果使用像 GrapheneDB 这样的提供者的 Neo4j，我们会觉得学习曲线变得平滑。

2.7 沙盒模式

Neo4j 提供一个非常方便的途径来尝试 Neo4j：沙盒模式。在 Neo4j 的官方网站上，点击几下鼠标，就可以在公有云上免费创建自己的 Neo4j 服务器。当然它是有限制的，那就是我们所创建的 Neo4j 服务器会在七天之后消失。但是，我们所做的所有工作都不会徒劳，因为我们可以下载沙盒的内容。听起来真棒，对吧？

除此以外，我们还能够在创建沙盒的时候，让我们的数据库默认附带上数据，如我们的 Twitter 账户中的推文数据等。

当前，通过沙盒模式创建 Neo4j 数据库我们可以得到如下数据：

- 特朗普政府相关的关系信息。
- 美国国会的投票信息。
- 电影、演员信息。

当然，利用沙盒模式我们也可以使用一个空白无数据的 Neo4j 数据库，方法如下：

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

步骤 01 现在让我们创建一个还有 Twitter 数据的沙盒，这需要一个 Twitter 账户。

步骤 02 在 neo4j.org 的主页（见图 2-19）上，单击 TRY NEO4J SANDBOX 按钮。

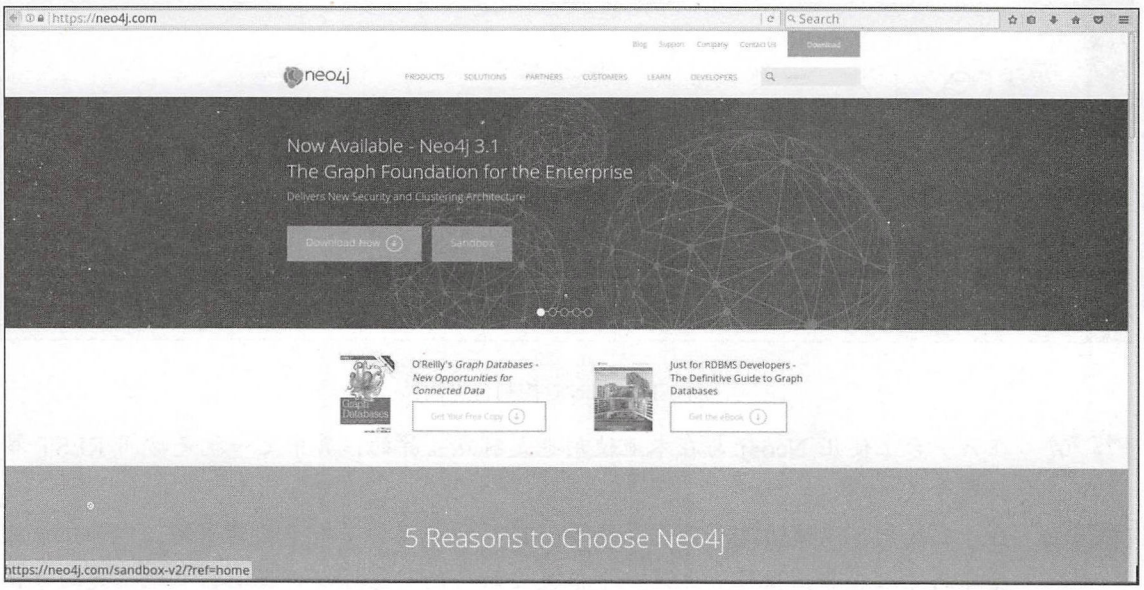


图 2-19 Neo4j 首页

步骤 03 下面我们不需要下载但需要一个稳定的网络连接环境，开始创建 Neo4j 沙盒模式，如图 2-20 所示。

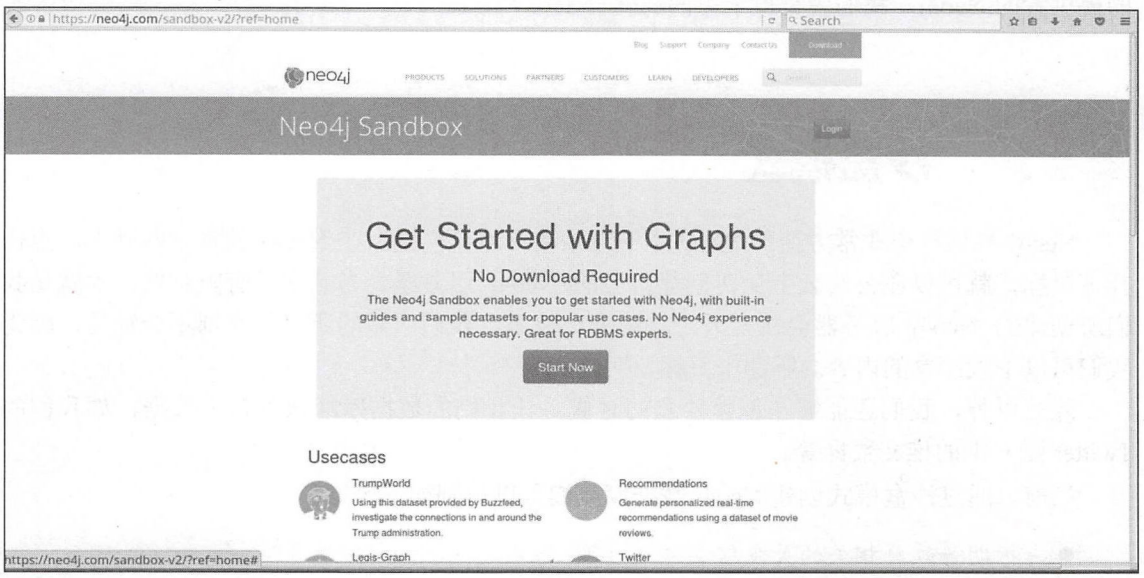


图 2-20 开始创建 Neo4j 沙盒模式

步骤 04 单击 Start Now 按钮后，没有要求我们登录而是弹出窗口，邀请我们通过我们选择的服务进行身份验证，如图 2-21 所示。

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

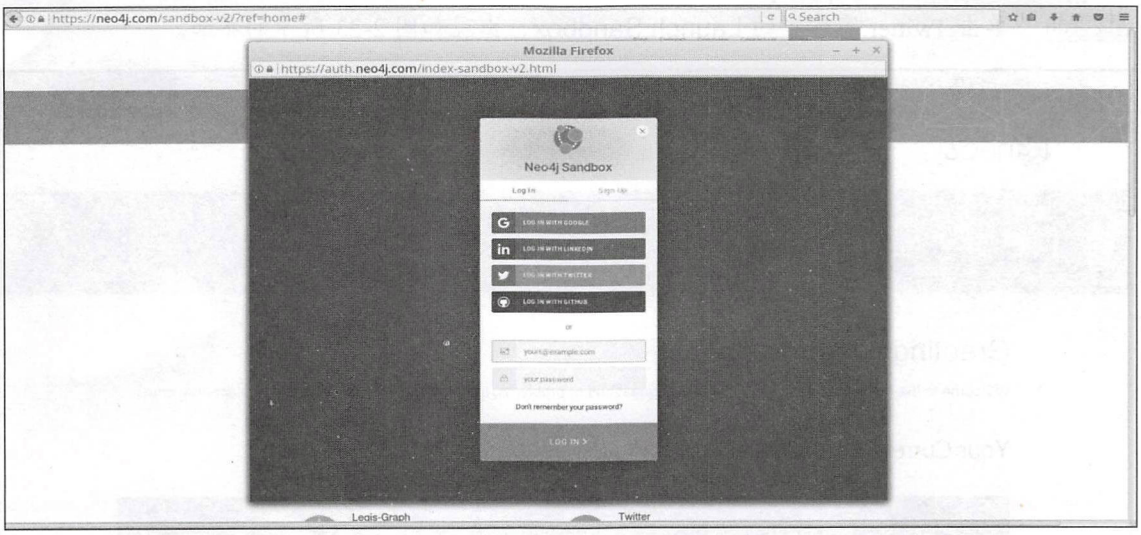


图 2-21　授权弹出框

步骤 05　在此，我们选择使用 Twitter 登录。接着会弹出另一个弹出窗口要求允许 Neo4j 访问我们的 Twitter 账户。下面我们会要求登录图 2-22 所示的页面。

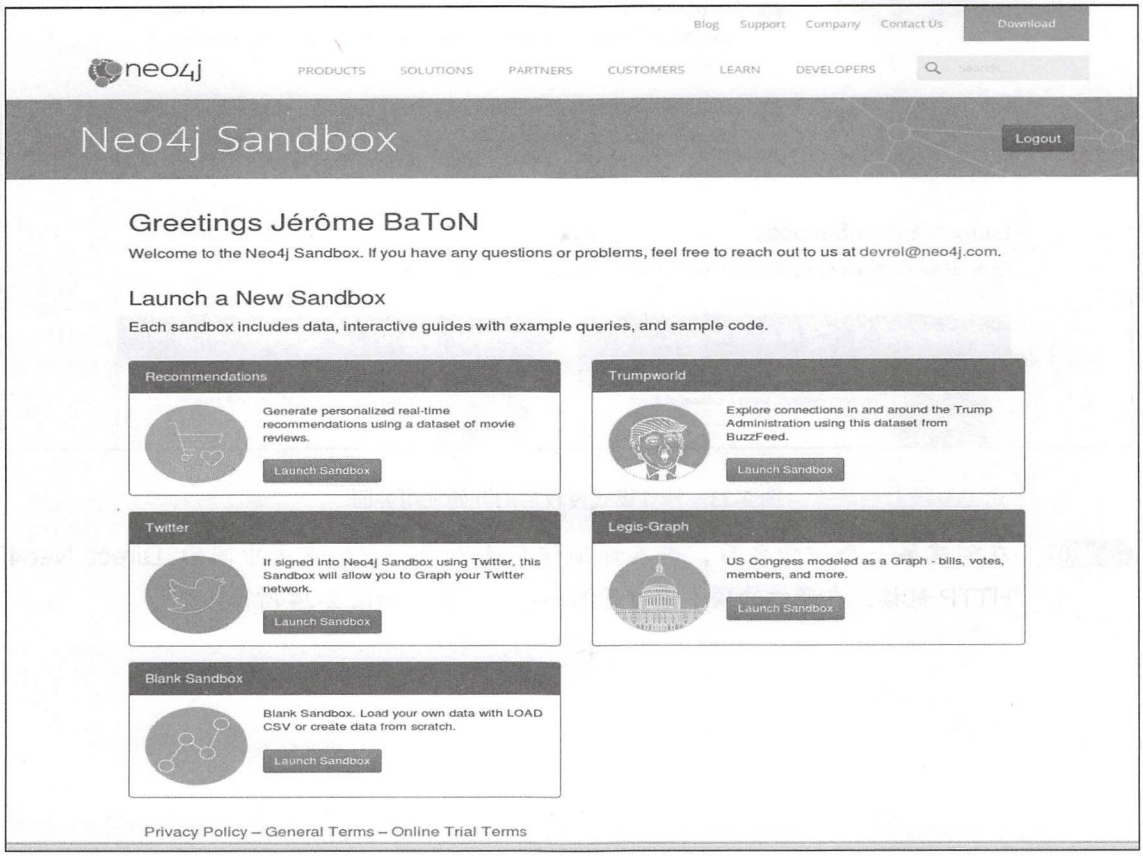


图 2-22　沙盒模式授权后的主页

非卖品，仅供非商业用途或交流学习使用
严禁网络传播本PDF，违者责任自负！

步骤 06 单击 Twitter 窗体中的 Launch Sandbox，出现如图 2-23 所示的界面。

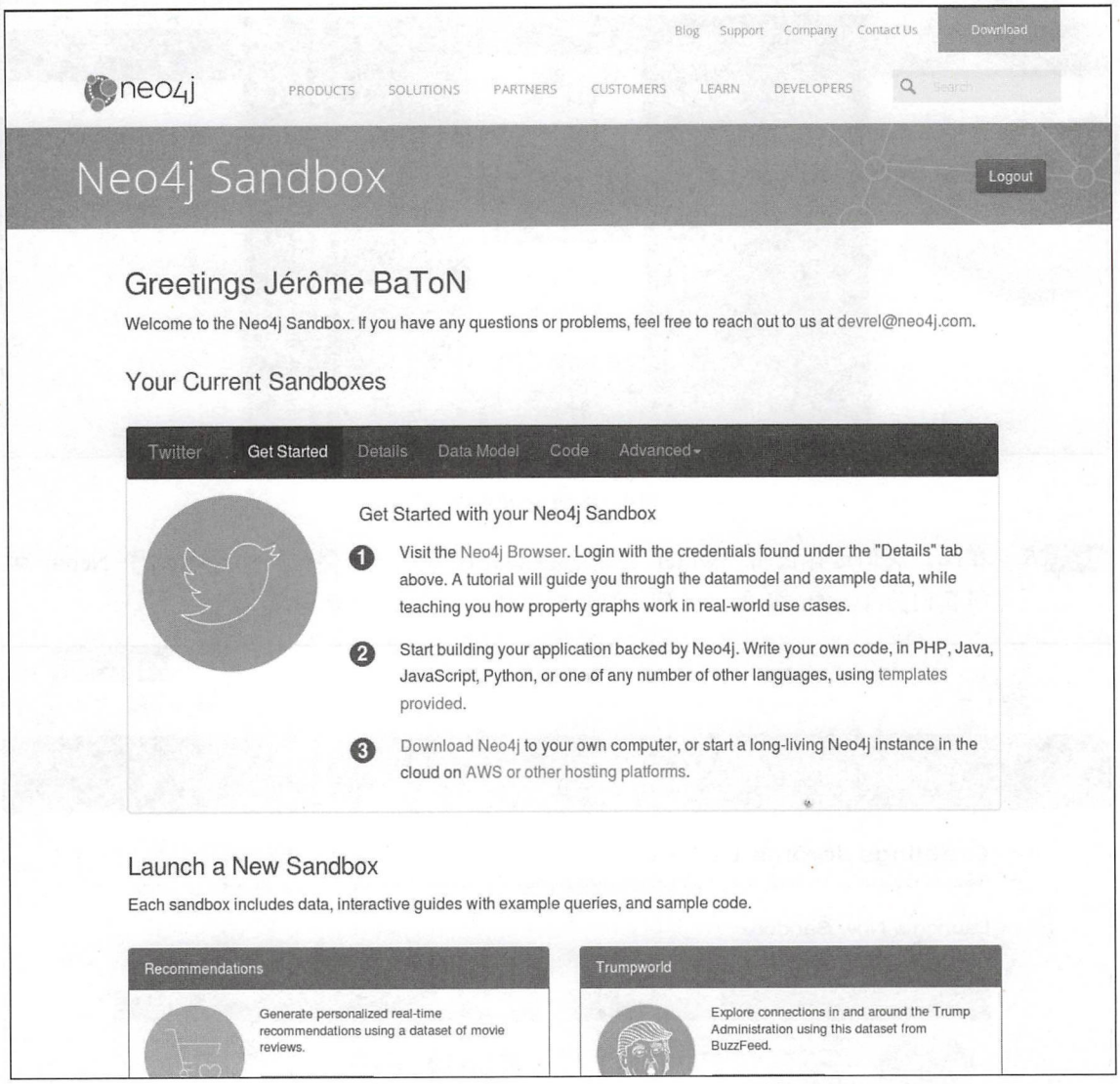


图 2-23 沙盒模式授权后的开始使用界面

步骤 07 在创建第一个沙盒之后，进入详细信息选项卡，我们点击中间的 Direct Neo4j HTTP 链接，在新的选项卡中打开 Neo4j 浏览器，如图 2-24 所示。

非卖品，仅供非商业用途或交流学习使用
严禁网络传播本PDF，违者责任自负！

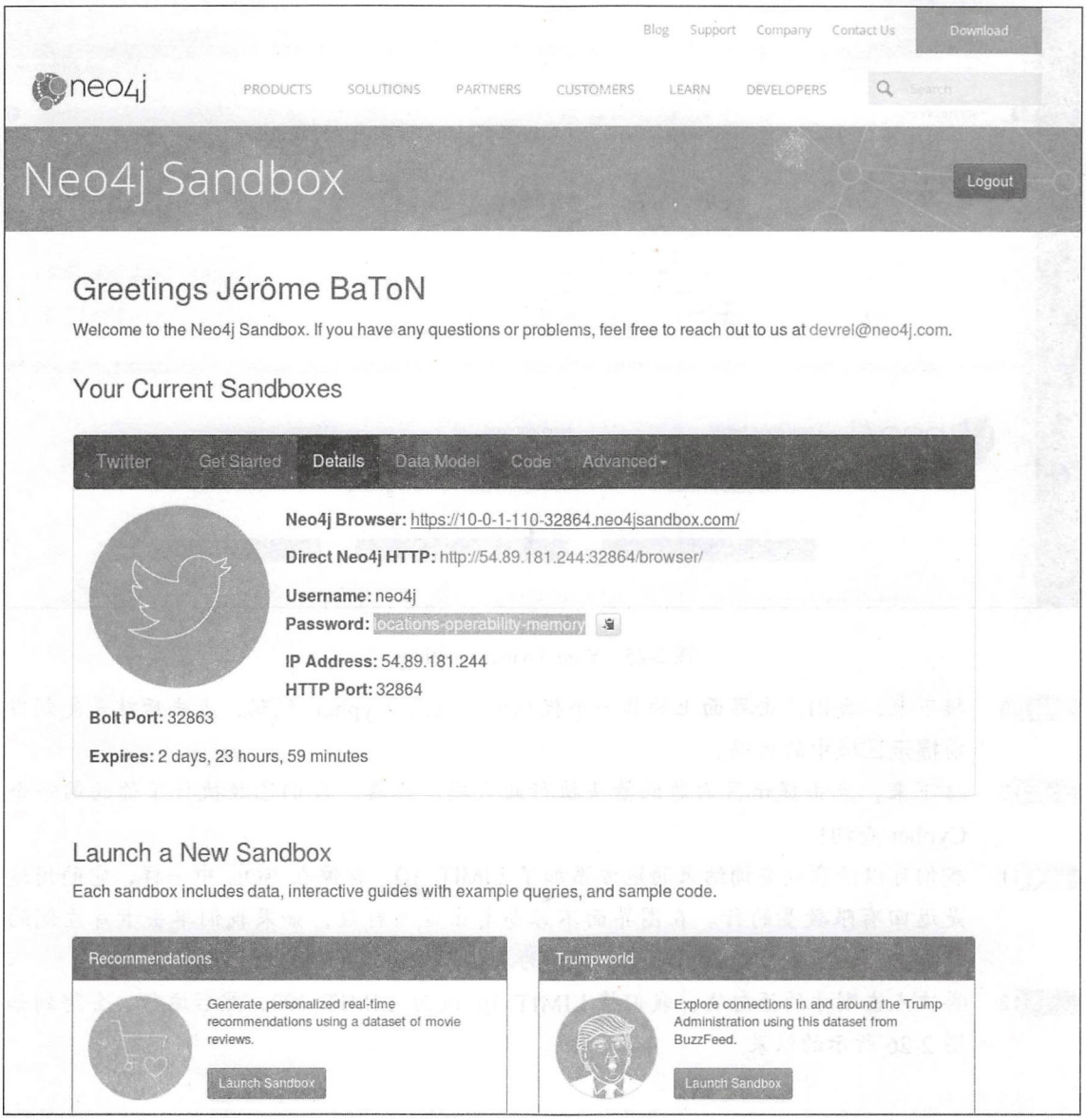


图 2-24　连接详情

步骤 08　卡片显示。点击右侧的右箭头两次，我们可以看到 Your mentions 标题，如图 2-25 所示。

非卖品，仅供非商业用途或交流学习使用
严禁网络传播本PDF，违者责任自负！

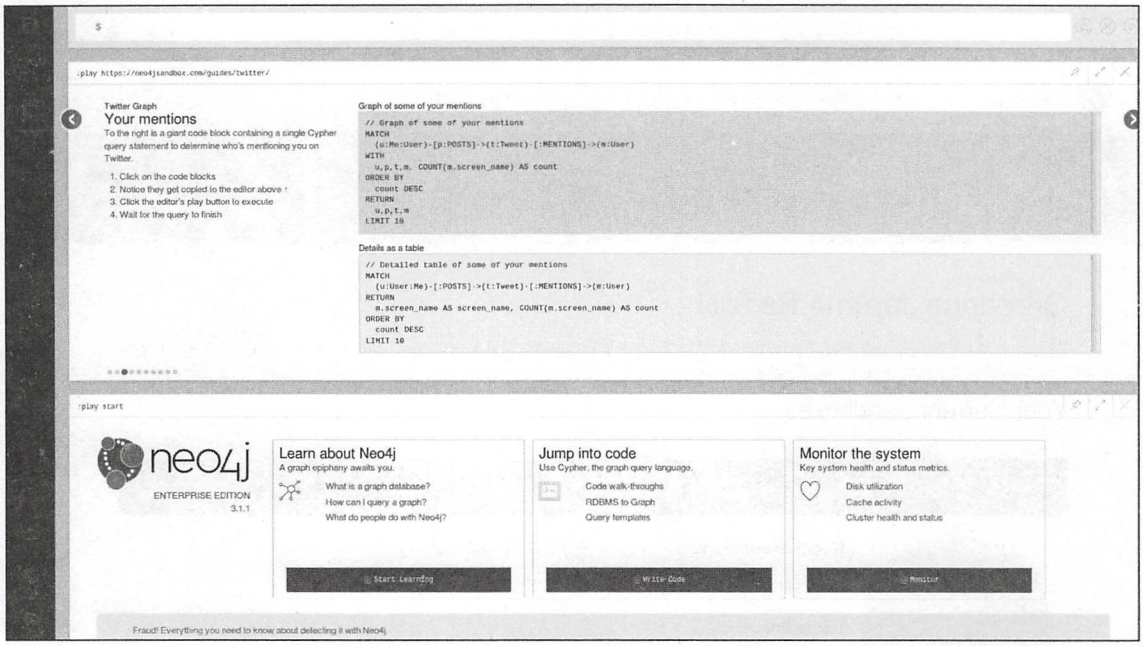


图 2-25　Your Twitter mentions

步骤 09　接下来，我们点击界面上的第一个代码块。这是 Cypher 代码，点击后就是复制当前**提示**区域中的代码。

步骤 10　接下来，点击提示区右边的箭头执行此代码。恭喜，我们已经执行了你的第一个 Cypher 查询！

步骤 11　我们可以注意到查询结束的地方添加了 LIMIT 10。就像在 SQL 中一样，它的用途是返回有限数量的行。在图界面不容易看出返回行数，如果我们单击卡片左侧的“**文本**”选项卡，则此图形可以显示为表格。你会注意到它有十行。

步骤 12　再次点击刚才的语句块，我们将 LIMIT 10 改为 LIMIT 300，然后运行，会得到如图 2-26 所示的结果。

非卖品，仅供非商业用途或交流学习使用
严禁网络传播本PDF，违者责任自负！

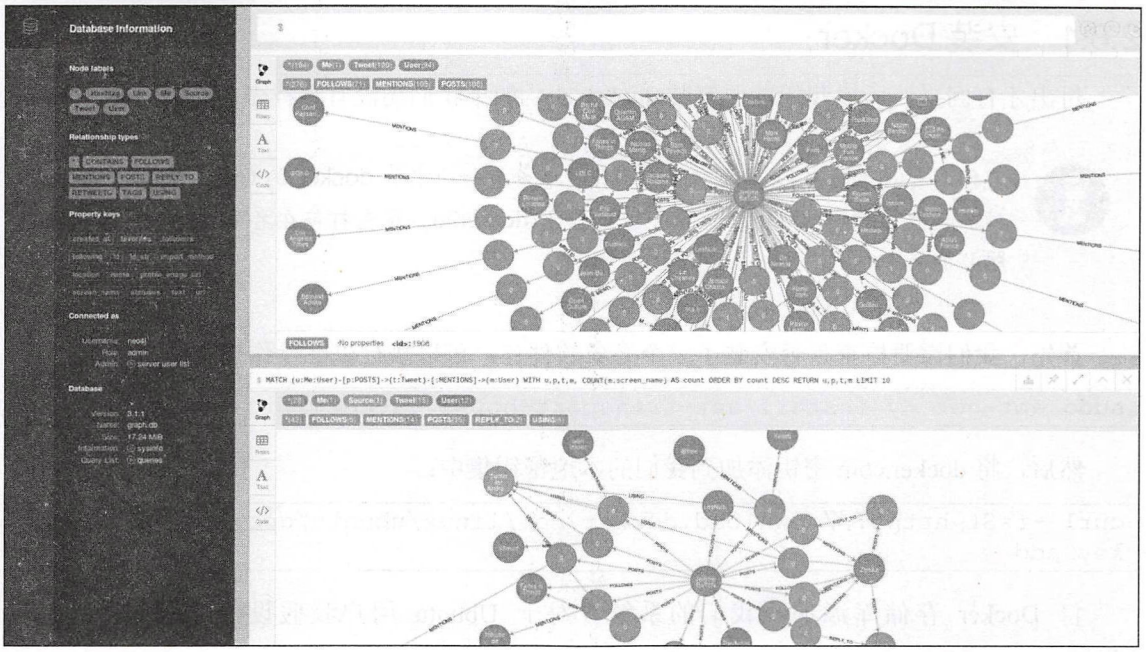


图 2-26 推文图数据

步骤 13　正如在图 2-26 中看到的，大量的节点堆积在一起降低了数据的可读性，即使我们拖曳某些节点依然很难理清。

注意，由于要考虑到浏览器的性能问题，对于大量的数据显示，Neo4j 控制台往往不能全部显示出来，但我们会收到一个警告，指出并非所有节点都已经显示出来。我们在浏览器中看到的内容可能始终是数据的局部视图。记住这一点。

2.8 在 Docker 容器中使用 Neo4j

如果我们想在本地安装 Neo4j，而不是把 Neo4j 直接安装在服务器上，并且我们不想面对沙盒的时间限制困扰，那么 Docker 来帮忙！

Docker 是一种虚拟化技术，可让我们在没有虚拟机的情况下运行系统映像。提倡学习使用这个技术，因为它允许我们做应用程序的**连续部署（CD）**。对于 Docker 相关的书籍，请在 Packt 网站上查看：https://www.packtpub.com/all?search=docker。

我们的目标是运行一个 Neo4j Docker 镜像，这样我们就可以像运行一个独立的服务器一样访问正在运行的镜像。对于这部分，我们将使用基于 Ubuntu 的 Linux Mint 社区版本。因此，我们的例子是针对 GNU / Linux 主机的。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

2.8.1 安装 Docker

付出才有回报，所以我们需要安装 docker-ce，如果我们还没有这样做的话。

我们不要对不同的 Docker 包产生混淆，曾经有一个名为 docker 的 Gnome 的系统桌面环境。但我们需要的 Docker 曾经被打包为 docker.io，现在打包在名为 docker-ce（社区版）的名下。

首先，我们需要检查是否安装了三个必备软件包，可以执行以下操作检查：

```
sudo apt-get -y install apt-transport-https ca-certificates curl
```

然后，将 docker.com 密钥添加到我们的本地密钥集中：

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-
key add
```

将 Docker 存储库添加到我们的系统（对于 Ubuntu 用户我假设你使用的是 64 位 CPU）：

```
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

如果使用 Linux Mint，则 lsb_release -cs 命令将不会返回支持 Docker 的 Ubuntu 版本名称（如 xenial 或 trusty），而会返回诸如 serena 或 sarah 的 Mint 版本名称。因此，Linux Mint 用户应该通过 https://fr.wikipedia.org/wiki/Linux_Mint#.C3.89ditions_standards 和 https://wiki.ubuntu.com/Releases 找到它们的等效 Ubuntu 版本，并将$（lsb_release -cs）替换为与之等效的 Ubuntu 版本。

2.8.2 准备文件

由于我们要跟踪日志并希望能够重用我们的数据，因此我们需要为 Docker 映像提供对文件系统的一些访问权限。在我们的 home 文件夹中，我们创建一个 Neo4j 文件夹和两个名为 log 和 data 的子文件夹。在 GNU / Linux 平台上我们用下面的命令就可以做到：

```
cd ~
mkdir neo4j
cd neo4j
mkdir logs
mkdir data
```

2.8.3 在 Docker 容器中运行 Neo4j

现在，在终端中运行下面这个长命令来运行一个带有 Neo4j 镜像的 Docker。

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

```
docker run --rm --publish=7474:7474 --publish=7687:7687 --
volume=$HOME/neo4j/data:/data \
--volume=$HOME/neo4j/logs:/logs neo4j:3.1.2
```

运行上面的命令后，系统会开始下载。因为我们的本地 Docker 存储库中还没有 3.1.2 版本的 Neo4j 镜像。

Neo4j 使用的端口分别是 7474、7473 和 7687，分别是协议 http、https 和 bolt 的端口。

在参数部分，你可以看到 volume 参数两次。它的用途是将本地文件系统上的文件夹链接到容器文件系统。

如果 Neo4j 默认的 3 个端口号事先没有被占用，那么我们的终端显示应该如图 2-27 所示。

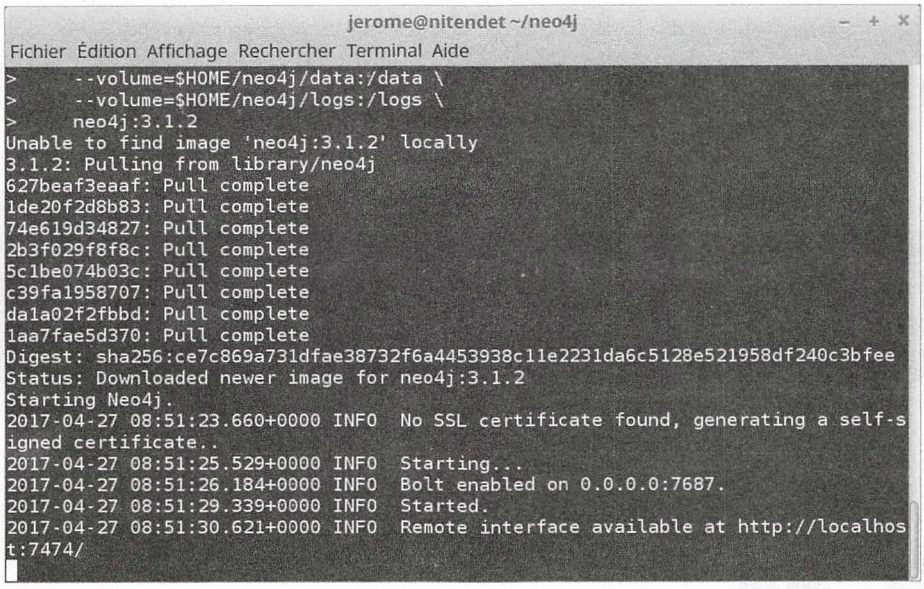


图 2-27　启动 Docker 容器

这告诉我们，Neo4j 希望你正常连接 7474 端口。这时，打开浏览器并打开 URL “http://localhost:7474”，我们可以看到熟悉的 Neo4j 控制台界面，至此我们安装成功了，我们可以操作 Neo4j 了。

如果要 Docker 停止运行我们的 Neo4j 镜像，那我们需要给出的不是镜像的名称，而是正在运行的容器的标识符（见图 2-28）。所以首先在另一个终端里输入如下命令：

```
docker ps
```

这将列出所有正在运行的容器，在我的例子中只有一个。我们看第一列 container_id 列，容器标识为“751be731234b”，我们将它作为参数：

```
docker stop 751be731234b
```


非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

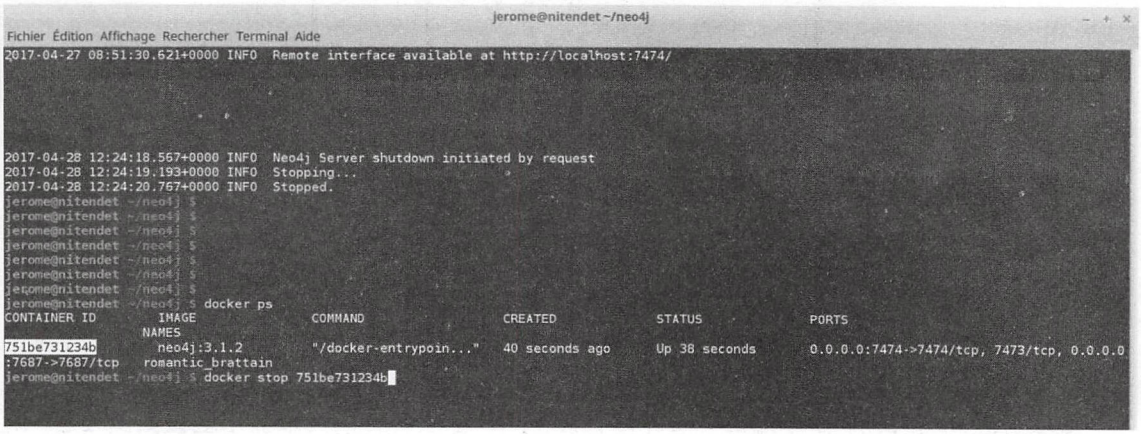


图 2-28　停止 Docker 容器

实际上，给定的参数只能是 ID 的一部分，只要能唯一标识出容器即可，在我的容器列表中只有一个容器，因此我只需要拿标识符的第一字符做参数也是可以的（见图 2-29）。

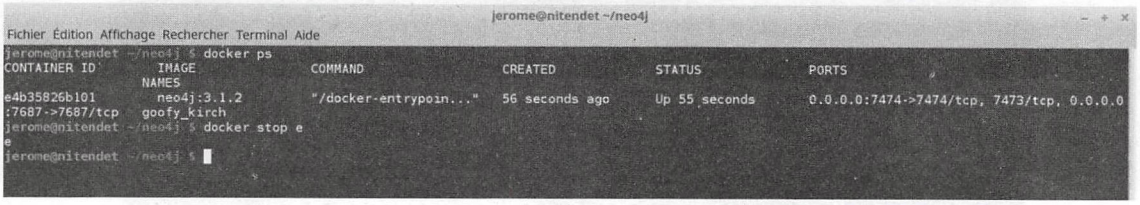


图 2-29　Docker 容器已经停止

2.9 问答

问题 1：Neo4j 是 ACID 数据库吗？

A. 是

B. 不是

问题 2：Neo4j 的企业版可以使用以下哪种许可形式？

A. Neo4j 是封闭的、专有的，别人需要从 Neo4j 官方购买

B. 开源许可形式：采用 Apache 2 许可协议

C. 开源许可形式：采用 Affero GNU Public 许可协议

D. 多重许可形式：要么从 Neo4j 官方购买，要么是开源的 Affero GNU Public 许可协议，要么是开源的 Neo Technology 商业许可证

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

问题 3：Neo4j 仅在基于 Linux / Unix / OS X 的系统上可用？

A. 是
B. 不是

问题 4：使用 Docker 意味着每次使用容器时都会丢失所有数据？

A. 是
B. 不是
C. 取决于所给的参数

2.10 小结

在本章中，我们讨论了 Neo4j 的背景，特别是它是世界领先的图数据库，其他内容还包括 Neo4j 图数据库概念的具体实现、使用案例、许可模式以及安装和部署的注意事项。现在我们已经有了足够的背景知识，能够充分准备好开始本书的动手部分了。这就是我们本章要达到的效果。下一章我们将讨论怎样用 Neo4j 进行数据库建模。

第 3 章

Neo4j数据模型

在本章中，我们将开始学习 Neo4j 的图数据库模型。

由于这种类型的建模与我们熟悉的关系数据库知识完全不同，因此我们先了解基础结构，然后继续探索一些常用方法。

本章将涉及以下主题：

- 四种基础数据结构
- 图模型 ——一种简单、高保真的现实模型
- 建模陷阱与最佳实践

3.1 四种基础数据结构

众所周知，图论为我们提供了许多不同类型的图。这些图具有不同的形状和大小，而 Neo4j 需要选择一种非常特殊的数据结构，这种结构要足够灵活，以支持真实数据集所需的多功能性。Neo4j 最终选择把标签属性图作为数据模型，它是所有图模型中最通用灵活的模型。

图数据模型（见图 3-1）为我们提供了四种不同的基础构建模块来构建和存储数据。

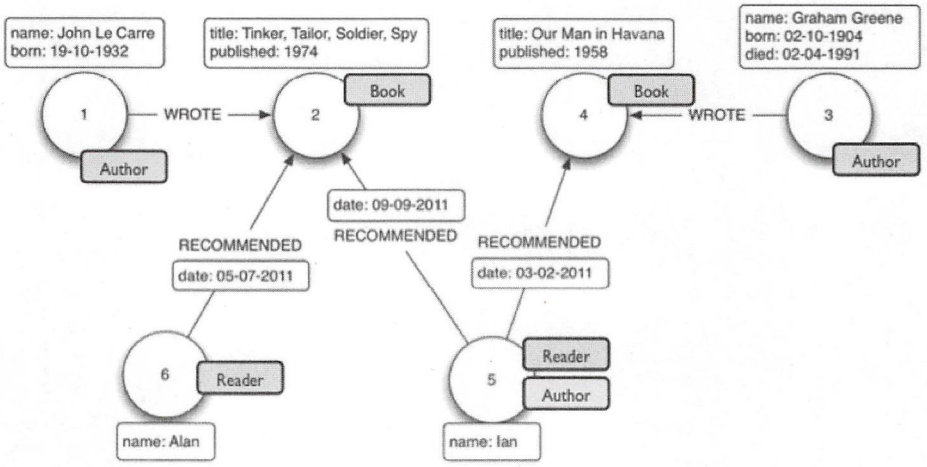


图 3-1　标签属性图模型

- **节点**：通常用于存储实体信息。在图 3-1 所示的图书馆数据模型中，节点是个人书籍、读者和作者。
- **关系**：用来将节点连接起来构建实体，它们相当于在关系数据库管理系统中显式存储和预先计算的连接查询。正如我们在前面的章节中看到的那样，连接不再是查询时的操作，它们就像遍历连接两个节点的关系一样简单。关系只能有一个类型，必须有开始节点和结束节点，以及指向。关系可以自我循环引用，但两头永远不能为空（缺少开始或结束节点）。
- **属性**：节点和关系都是属性的容器，它们实际上是键值对组成的。对于节点，非常容易理解，和关系数据库记录一个或多个字段或属性一样，节点也可以拥有一个或多个属性。对于关系，也可以有属性（如连接表中的附加列）。属性用于进一步强化关系，并且可以在查询、遍历期间用于评估我们正在查找的模式。
- **标签**：2013 年底新增至版本 2.0 的 Neo4j 基础数据模型结构。使用标签能够快速高效地对节点分类并创建子图。通过给节点分配标签，Neo4j 使得大多数用户的数据模型变得更加简单。节点不再需要使用类型属性，或者将节点连接到提供关于图的元信息的定义节点。Neo4j 现在已有的这些特性，给我们带来了极大的帮助。在编写本书时，标签主要用于索引和一些有限的模式约束。将来，标签提供的有关存储在图中的数据结构很可能会被用于其他目的，如附加模式、安全性、图形分片、分发等。带有标签的节点等同于关系数据库表中的行数据。

Neo4j 允许我们为每个节点设置多个标签，这在关系范式中没有语义等价。
关系只能有一个标签。
有了这四种数据结构，我们现在就可以开始使用 Neo4j 了。

3.2 图数据库建模基础

在本节中，我们先花一些时间来探讨图数据库模型。具体而言，我们先解释一个熟悉的关系数据库系统知识中的常见误解。

3.2.1 ER 图和关系模式

在关系系统中，我们通常用实体——关系图开始建模。可以从问题、域描述（在当今敏捷开发方法中称为用户案例）入手，并抽象出实体和关系。我们稍后讨论这个问题，但实际上，我们通常从这样的域描述中发现，我们可以执行以下操作：

- 通过名词描述提取实体。
- 通过形容词描述提取属性。
- 通过操作动词描述提取关系。

当然，这些都是通用的准则，需要在每个领域单独进行尝试和测试，以确保它是合适的。现在我们来看图 3-2。

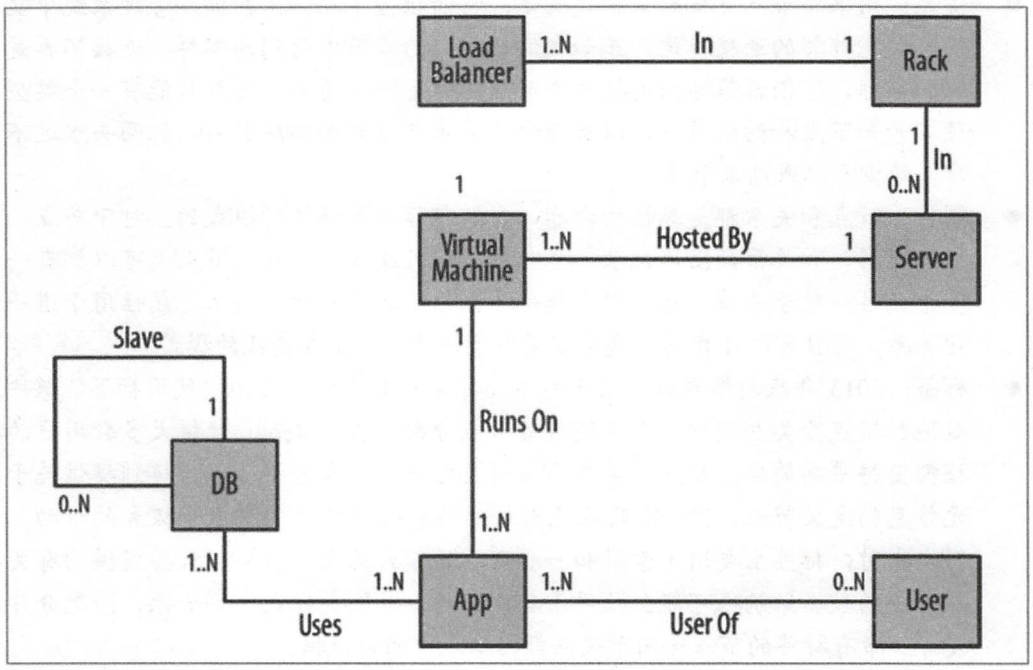


图 3-2　实体关系图

从图 3-2 可以看出，ER 图至少在现实世界模型中捕获业务领域有独特的优势，但是，也有不少缺点。

尽管视觉上已经接近图可视化，但是在捕捉丰富业务领域上，ER 图即刻显示出了关系模型的不足。尽管允许关系被命名（图数据库完全支持，但是关系数据库没有），但是 ER 图只允许单个、无向、已命名但实体之间没有限定的关系。

在这方面，关系模型不适合描述实体之间关系众多、语义丰富且多样化的现实世界。正如我们之前所看到的，带标签的属性图允许对领域进行更加丰富的描述，特别是关于实体之间的关系，这些关系将通过属性被多重化、定向化和限定化。

当我们将 ER 图应用到实际系统中时，关系型 ER 模型的问题变得更加严重，并且有很大局限性。让我们来看看，当我们在现实世界进行测试时，关系模型的基本问题是如何暴露出来的。

3.2.2　通过连接表引入复杂性

把之前描述的模型交给数据库管理员来实现。那么会发生什么呢？在实现过程中，关系模型会非常复杂。从图 3-3 中我们看到，实际上每个关系都有 n-n 种组合，我们需要连接两张表。

这就是我们所说的**连接表**，用于请求 n-n 种实体组合的查询。

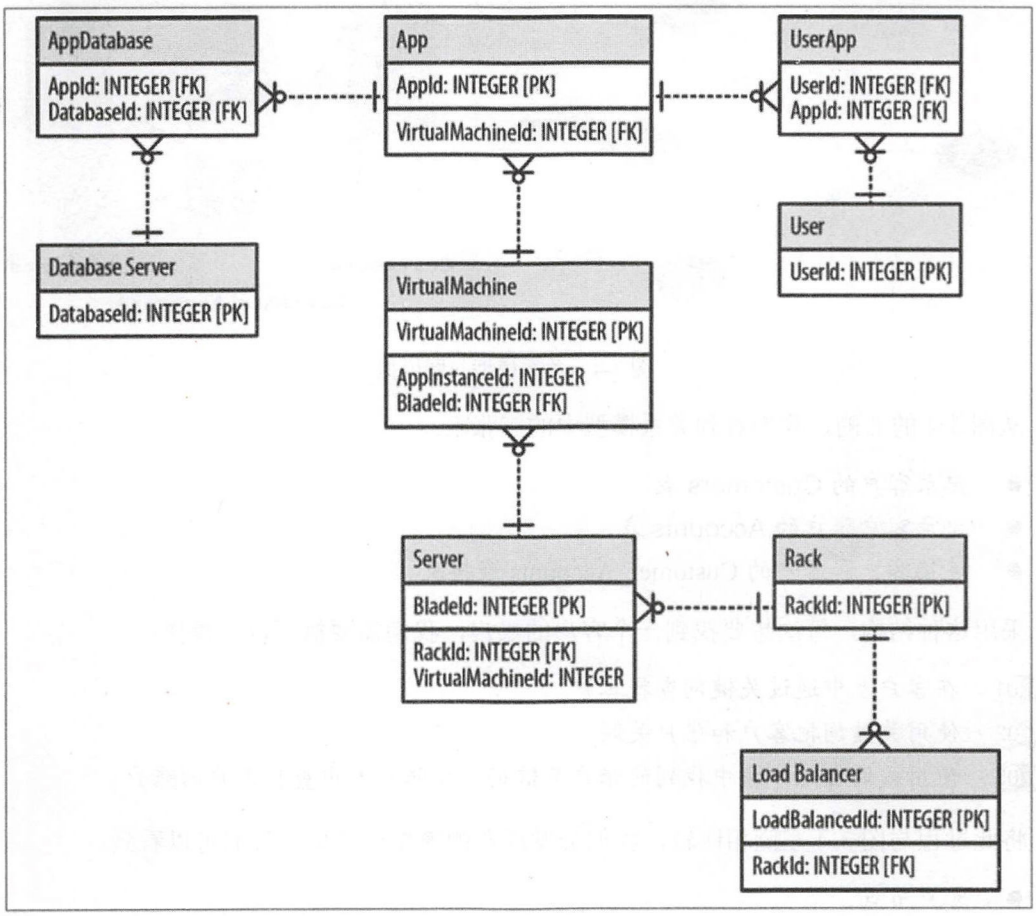


图 3-3　数据库模式

在前面的示例中，我们引入 AppDatabase 表来将应用程序连接到数据库服务器，并引入 UserApp 表来将用户连接到应用程序。这些连接表是为了解决关系模型的缺点，却使数据库管理员和应用程序开发人员的工作变得复杂，并且引入了不必要的复杂度。

3.3 图模型 —— 一个简单、高保真的现实模型

让我们快速浏览一下如何避免以前在图形世界中提到的复杂性。在图 3-4 中，我们将找到图模型和关系模型的对比。

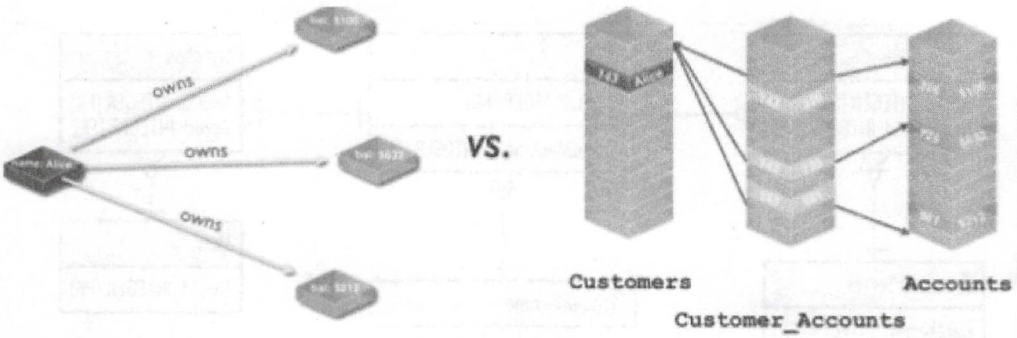


图 3-4　关系模型与图模型

从图 3-4 的右侧，我们看到关系模型中的三张表：

- 记录客户的 Customers 表。
- 记录客户账户的 Accounts 表。
- 连接客户和账户的 Customer_Accounts 连接表。

采用这种结构，每次想要找到一个客户的账户，我们需要执行以下操作：

步骤 01　在客户表中通过关键词查找客户。

步骤 02　使用关键词把客户和账户关联。

步骤 03　使用我们在上一步中找到的账户关键词，在账户表中查找客户的账户。

将此过程与图 3-4 左侧相比较，我们会发现左侧模型更简单。我们可以看到：

- 客户节点。
- 三个账户节点。
- 关联账户与客户的三种关系。

查找客户的账户过程非常简单，执行以下操作：

- 根据预先指定的关键词，通过索引查找客户。
- 从客户节点开始遍历，找到“owns”关系的账户节点

前面的例子，我们仅对两张表执行连接操作。随着连接操作的增加，操作难度在关系数据库管理系统中将指数级递增。随着参与连接操作的数据集越来越多，操作会越来越消耗大量系统资源。计算两个集合的笛卡尔儿积（这是关系数据库为了执行连接而需要做的事情）会随着表逐步增大而变得越来越复杂。

通过与关系模型比较，我们希望能对图建模有了初步认识，下面我们继续讨论一些陷阱和最佳实践。

3.4 图建模——最佳实践和陷阱

在本节中，我们将概述图数据库建模的一般性建议和最佳实践，同时也会提供一些洞见，以帮助我们避免常见的陷阱。

不言而喻，所有这些建议都是通用建议，在我们的特定领域中可能会有这些规则的例外情况 —— 就像以前一样，在关系数据库设计模型的情况下也是如此。

3.4.1 图建模最佳实践

在接下来的章节中，我们将一起探讨在 Neo4j 项目中成功应用的案例。

1. 查询性能设计

与任何数据库管理系统一样，尤其对于 Neo4j 这样的图数据库管理系统更是如此，查询驱动模型。也就是说，和以前使用过的或现在仍在使用的任何类型数据库一样，我们需要依据特殊考虑以做出特别的设计决策。因此，即使像 Neo4j 这样的图数据库，也没有一个完美的方法来建模。这一切都将取决于我们的数据，数据驱动我们的设计和模型。

因此，图数据库建模最重要的最佳实践，就是从用户案例开始的。用户想从我们正在设计的系统数据中知道什么？案例的例子可能是这样的：

“作为一名员工，我想知道在我工作的公司里有谁与我有相似的能力，以便我们可以交流知识。”

这段摘录告诉我们模型中的实体以及实体之间存在的关联。不同的领域描述可能会添加相似或不同的实体，以及相似或不同的关联，逐渐完善模型。

2. 用例关系匹配

考虑查询性能，并且能够让查询驱动模型，我们在建模时通常使用不同的关系类型，不同的节点使用不同的用例。许多大型的图数据库模型，针对不同的应用场景，同类节点使用多种关系。

为什么推荐的最佳实践在开发工作中实际应用的一个原因是研发支出（作为开发人员的价格，主要是增加模型复杂性）为特定用例引入两个节点之间的特定关系，实际上是如此之低。没有附加的表或模式要创建，具体来说，不需要计算额外连接，图遍历只是需要使用不同的路径来建立跨数据库存储的网络。

将关系与用例匹配时要牢记的关键概念是关系类型的命名策略。一般来说，应尽量少地使用通用名称，例如 HAS_A 或 IS_PART_OF，而用能具体描述这些命名含义的名称。

3. 查找 n 元关系

我们第一次阅读用户案例有时并不会得到最佳结果，这可能有很多原因，但通常是因为我们没有发现案例中隐藏的模型元素。

在处理所谓的 n 元关系时经常出现这种情况。当我们想要关联两个以上的事物时，关系

类型往往隐藏在模型中，在某些情况下，最简单有效的方法就是将原型通过关系连接起来。这些服务于两个（实际是 n 个）事物或原型的关系被称为 n 元关系。这是一种非常通用的建模模式。

当在图模型中发掘出新的关系类型时，意味着我们发现新的节点，能够分离出新的实体，如图 3-5 所示。

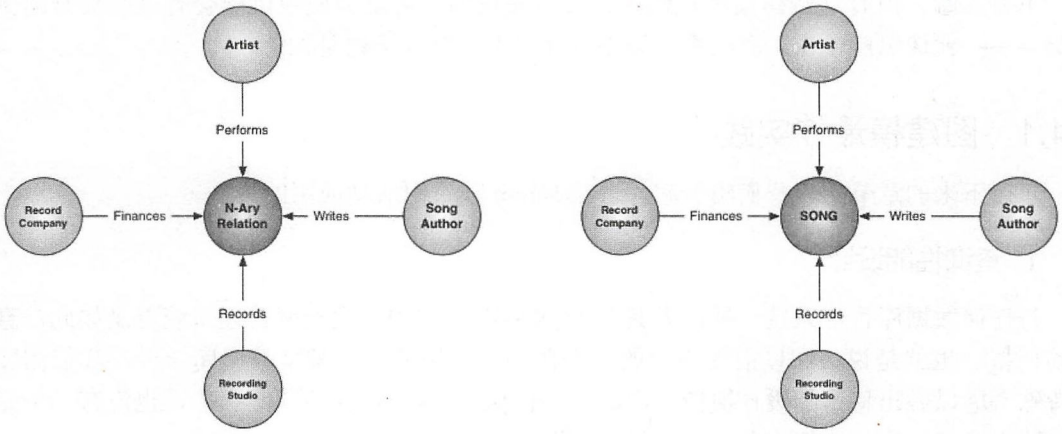


图 3-5　将 n 元关系转换为节点

这正是我们在前面的例子中所做的。

4. 节点粒度

在本节中，我们将要探讨的典型图建模模式被称为**粒化模式**。在图数据库建模时，倾向于使用比我们习惯的关系模型具有更高粒度、更精细的数据模型。

在关系模型中，我们遵循**数据库规范化**的过程来提出模型的粒度。维基百科定义该过程如下：

“……组织关系数据库的字段和表以减少数据冗余并提高数据完整性的过程。规范化通常涉及将大表拆分成小表（不是冗余），并定义它们之间的关系，目的是隔离数据，可以在一个表中添加、删除和修改字段，然后把定义的关系传递给数据库的其余部分。”

实现的过程是创建越来越小的表结构，直至达到第三范式。众所周知，在 IT 行业：一旦数据库达到第三范式，就认为数据库已经满足要求了。你可以访问 https://en.wikipedia.org/wiki/Database_normalization#Normal_forms 了解更多细节。

正如我们之前所讨论的，这个模型因为引入了对查询时连接表和连接操作的需求，实现代价比较高。数据库管理员倾向于数据非规范化，但会导致数据重复，这是另一个非常棘手的问题。

然而，在图数据库建模中，归一化使得连接操作更容易执行。这就是为什么我们在图模型中倾向于创建瘦节点和关系，也就是节点和关系有尽量少的属性。这些节点和关系非常细微，并且已经被粒化。

我们每次建模时都要问自己关于这种模式的一个特别问题——应该把它作为属性还是节

点呢？ 例如，我们是否应该将啤酒的酒精百分比建模为啤酒品牌的一个属性？图 3-6 展示了酒精百分比作为属性的模型。

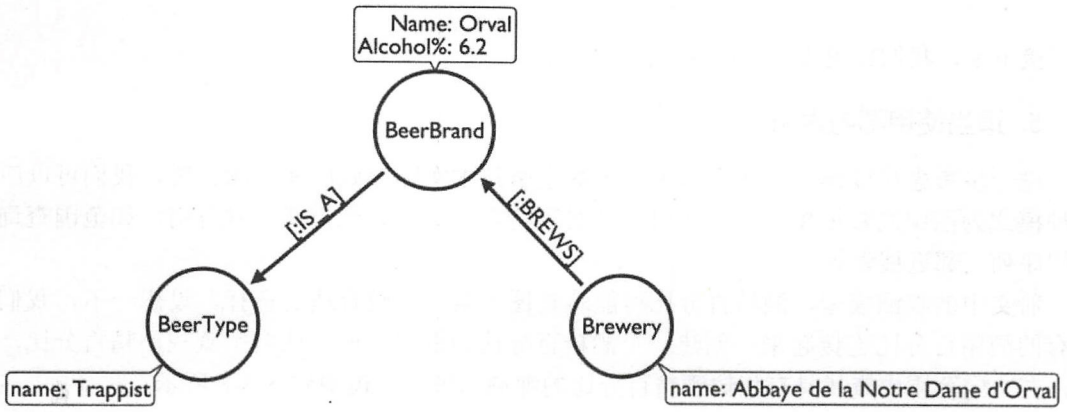


图 3-6　胖节点的数据模型

另一种方法是将酒精百分比拆分成不同的节点，请看图 3-7。

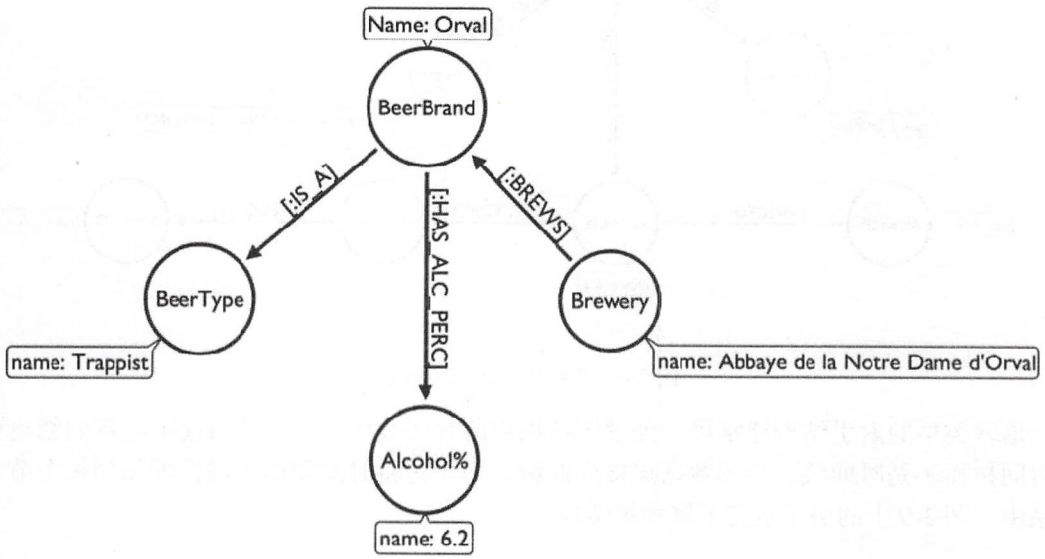


图 3-7　节点粒度结构的数据模型

哪一种模型正确？都不是。我们应该根据查询来确定哪种模型是适合的。一般来说，我会提出以下论点：

- 在图遍历过程中，如果不需要评估酒精百分比，我们会把它作为遍历的最终节点的属性。毕竟，这样做让我们的模型更简单，每个人都倾向简单。

- 在图遍历过程中，如果需要评估特定（一组）啤酒品牌的酒精百分比，那么将其分离成单独的节点类别可能会更合适。遍历节点通常比评估每个路径的属性更简单快捷。

接下来，我们将更进一步，使用图内索引。

5. 适当使用图内索引

进一步考虑粒度模式，我们知道大多数索引技术实际上底层使用图、树，我们可以应用这种模式为图中的数据模型创建索引。这对于特定类型的查询模式非常有用，如范围查询、时间序列、邻近搜索等。

前文中的啤酒模型，酒精百分比可能就是图内索引比较合适的选择。设想一下，我们将所有的酒精百分比连接起来，创建一个酒精百分比的链表，向上或向下查找酒精百分比，例如，在一定范围内查找具有相似酒精百分比的啤酒品牌。该模型如图 3-8 所示。

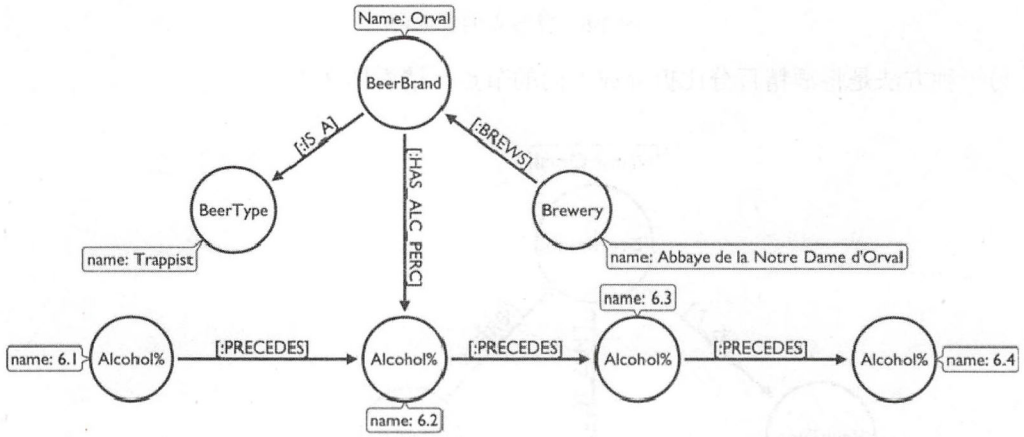


图 3-8　酒精百分比的图内索引

这些类型的索引结构通常用于处理图结构中的时间数据。在这种情况下，我们创建更多的时间树而不是时间线，将图数据连接到此树，而不是将时间戳作为属性添加到每个节点或关系中。图 3-9 中的例子包含了这种树结构。

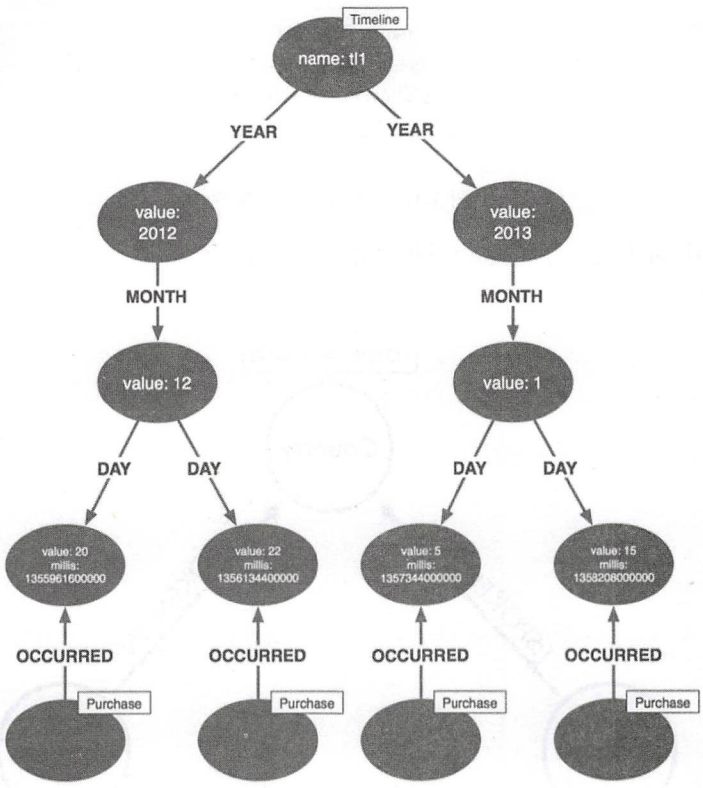


图 3-9　图内时间树索引

图 3-9 中的所有模式都是成功应用于项目中的常用建模模式。聪明地使用它们，牢记它们都是关于查询模式的。了解用户的真实需求将会大大影响它的设计模型，而且我们可能需要并且想要多次迭代该模型以使其达到稳定的状态。幸运的是，像 Neo4j 这样的图数据库管理系统非常善于处理这种变化，并且允许我们做到这一点。

3.4.2　图数据库建模陷阱

与任何类型的数据库建模一样，图数据库也会有一些我们应该尝试的缺陷。本节不会试图给出一个详尽的清单，但是应该给出一个可能导致建模决策不佳的实践类型。

1. 使用丰富的属性

因为在图内数据库系统中有一个非常精细的模型实际上是一个最佳实践，与此相反，这往往是一个反模式。使用表面上过于丰富（如图 3-10 所示）的节点属性，可以通过利用模型，将属性拆分为单独的节点，并使用关系连接它们来更好地解决。

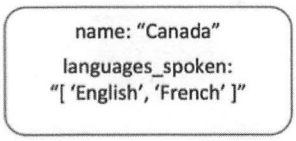


图 3-10　使用丰富的属性

请看图 3-11 中的图表，了解这种反模式的潜在解决方案。

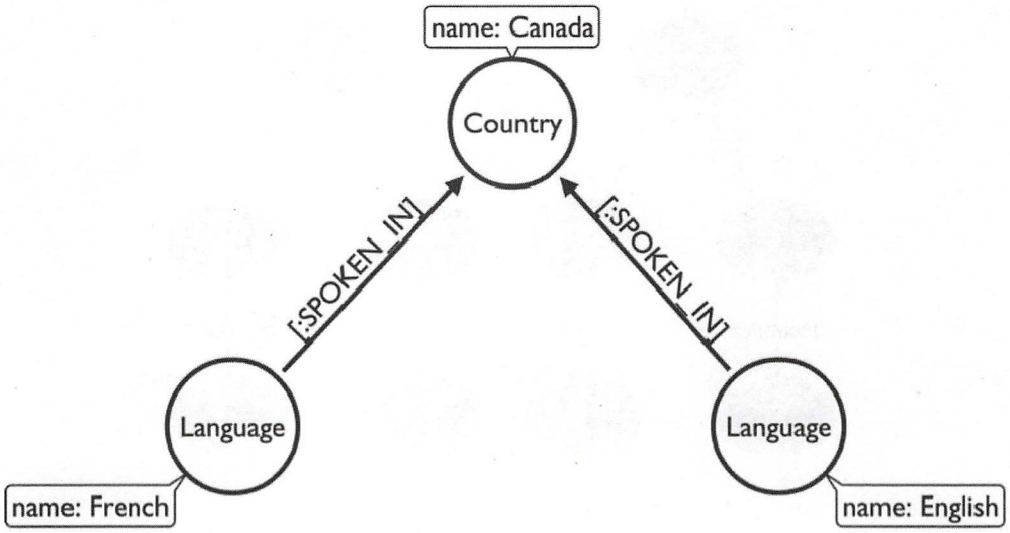


图 3-11　造粒丰富的属性

2. 多概念节点

我们所看到的另一个反模式是多次引入的，就是不同的关注点或概念在模型中没有被恰当地分开。在图 3-12 所示的模型中，我们将国家概念、语言概念和货币概念混合在一个节点结构中。

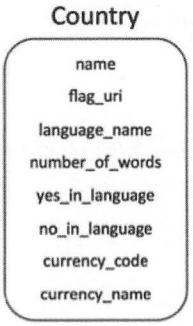


图 3-12　一个富裕的国家

这应该是一个危险信号，因为它将不可避免地在查询时给我们带来困难或限制。把概念分解成不同的国家、语言和货币节点结构（见图 3-13），每一个都有自己的特点和属性，更加明智，这正是我们试图传达的。

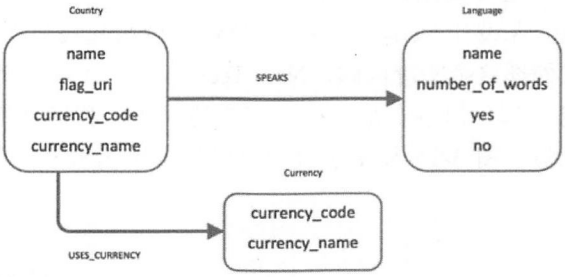


图 3-13　一个更丰富的模型

3. 未连接的图

图建模反模式的一个明显例子是非连通图。图是实体间的连接，图数据库的功能则是遍历这些节点间的连接。

图 3-14 中显示的模式不应该在图数据库中使用。

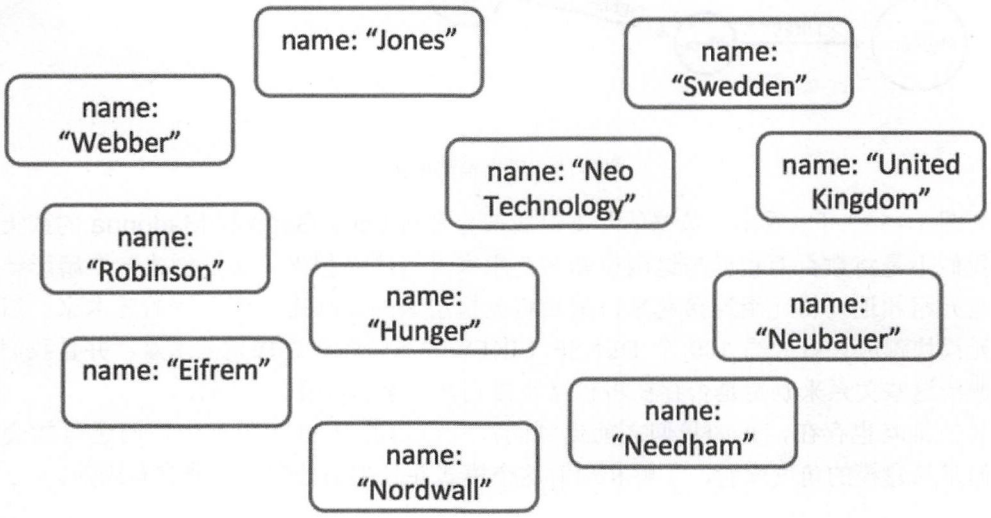


图 3-14　非连通图

关系在图中提供结构和查询能力，所以我们应该充分利用关系。

4. 密集的节点模式

我们已经讨论了图查询如何在图数据库系统中工作，基本原理是图本地查询：从一组明确的起点开始查询数据，遍历速度通常非常快，并且仅取决于查询将访问的图的数量。通常情况下，随着数据集大小的增加，遍历速度也会非常稳定，因为在一个普通的图模型中，任

何东西都没有相连。查询将只评估连接到起点的图形部分，因此，遍历速度将是顺畅的。

数据集中发生的一个非常有趣的问题是图的某些部分全部连接到同一个节点。这个节点也被称为密集节点或超级节点，成为图遍历的一个真正的问题，因为图数据库管理系统将不得不估计到该节点的所有连接关系，以确定下一步将在图遍历。在图数据库模型中，超级节点会是一个棘手的问题，在设计 Neo4j 实例时应该避免使用超级节点。

存在不同的策略来处理这个密度问题，Neo Technology 正在对 Neo4j 做出一些重要的改变，使这个问题更容易处理。但事实是，在大多数情况下，我们都希望通过使用合适的模式来避免碰到这种技术风险。图 3-15 显示了我们可以从音乐艺术家、粉丝图形网域使用的一个潜在策略。

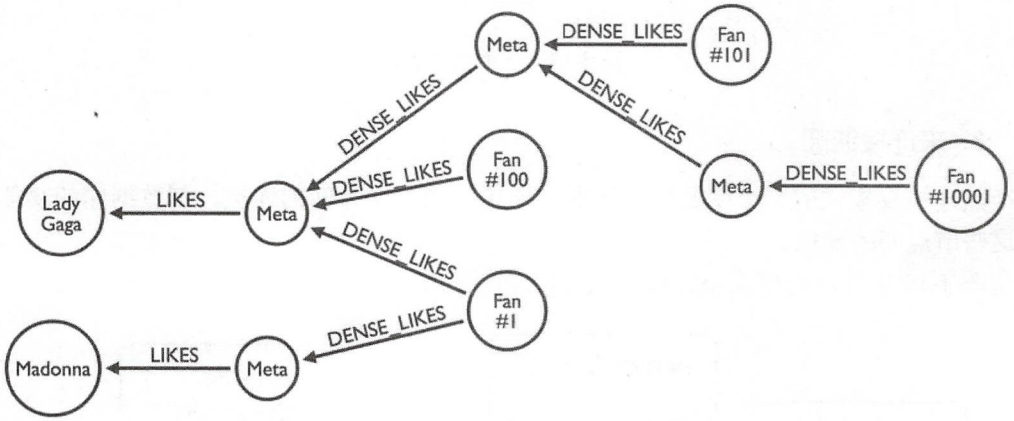


图 3-15　扇出策略密集节点

从图 3-15 中可以看出，策略的基本组成部分是将 Lady Gaga 和 Madonna 的密集关系分开。我们不是将数百万粉丝与这两位知名艺术家直接联系起来，而是创建一个扇形结构，将粉丝与元组相连，将元组连接起来，最后将元组层级的顶部连接到实际的艺术家。然后建议每个元模块都应该有大约 100 个 DENSE_LIKES 关系将粉丝连接到艺术家，并且我们可以快速地遍历这些关系来确定是否存在将粉丝连接到艺术家的可变长度路径。

其他策略也存在，正如我们之前提到的，在以后的 Neo4j 版本中，问题可能会大大减少，但是从建模的角度来看，了解和理解这个模式并且主动处理它是非常有用的。

3.5 问答

问题 1：Neo4j 的四种基本数据结构是什么？

A. 表、记录、字段和约束

B. 节点、关系、属性和模式

C. 节点、关系、属性和标签
D. 文档、关系、属性和集合

问题 2：图数据库模型的标准化非常困难。

A. 是
B. 否

问题 3：如果数据集中有几个与其他实体有很多关系的实体，则由于密集节点问题而无法使用图数据库。

A. 正确，你将不得不使用关系系统
B. 确实，但是别无选择，所以你必须忍受
C. 错误，你仍然可以使用图形数据库，但是对于所有查询来说，这将会非常缓慢
D. 错误，你可以非常有效地使用图形数据库，但是你应该采取预防措施，例如对数据应用扇出模式

3.6 小结

在本章中，我们讨论了如何为图数据库管理系统建模。我们谈到了模型的基本组成部分，并将其与我们在关系数据库管理系统使用的方式进行了比较和对比，然后讨论了建模过程中经常出现的适合的和不适合的模式。

众所周知，建模过程最好是邀请非技术人员，和团队成员一起在白板前完成。

有了这些模型，我们现在可以开始学习更多的知识，如 Neo4j 的查询语言：Cypher。

第 4 章

◂ Cypher入门 ▸

Cypher 是 Neo4j 的**查询语言**（Query Language, QL）。数据库系统需要查询语言以方便人和软件应用与之高效地交互。当前已经有一些图数据库查询语言，如 Gremlin、GraphQL 和 SparQL 等，其中的一些在一定程度上还激发了 Cypher 的创造，但 Cypher 与之前的所有查询语言有很大的不同。可能你看过有关 Neo4j 的书籍、演示文稿和课程，但没有哪个是完整的，都只涉及其中的一部分。当然，我们这里不仅仅是入门。

本章包括以下内容：

- Cypher 的语法
- Cypher 的关键特性
- 与数据很好结合
- Cypher 的关键操作词
- 语法约定
- 高级知识
- Cypher 参考卡
- openCypher 工程

4.1 Cypher 的语法

我喜欢用一些简单的例子来解释问题，就让我们以这种方式开始吧。如果我们想用 Cypher 来表达罗密欧（Romeo）爱朱丽叶（Juliet），那么 Cypher 的语法如下：

```
(romeo:Person{name: "Romeo"})-[:LOVES]-
>(juliet:Person{name:"Juliet"})
```

这几乎就是 ASCII 艺术（ASCII-Art）。看到这个模式了吗？

```
(NODE1)-[:RELATION]->(NODE2)
```

现在你应该想在你的数据库中创建这些节点及其关系了。在 Neo4j 的浏览器（如果你已启动 neo4j 服务器，可通过 http://localhost:7474 访问）上面的提示框中输入如下语句：

```
CREATE (romeo:Person{name: "Romeo"})-[:LOVES]-
```

```
>(juliet:Person{name:"Juliet"})
```

这时，我们将得到一条消息，告知我们创建的节点和关系的数量。

我们刚刚已经完成了数据创建。你刚才是否说了“哇”。是的，它就是这么简单。的确，我展示 Cypher 给每个人的时候都令他们印象深刻。一些学生甚至觉得很惊讶！

当我们刚刚将数据插入的是一个新数据库时，如果我们想看它可使用如下查询：

```
MATCH (n)
RETURN n;
```

它将返回所有的节点（和它们的关系）。现在，让我们回过头来看看 Cypher 的历史及理论知识。

4.2 Cypher 的关键特性

在创建 Cypher 时，Neo 公司和 Andres Taylor（https://twitter.com/andres_taylor）致力于**创建一种新的查询语言，专门用于处理存储**像我们存放在 Neo4j 中的这类**图数据结构**。这样做有很多种原因，但更重要的是，Cypher 有如下四个其他查询语言中没有的特性。

让我们快速看一下这些特性，它们非常重要，能够帮助我们理解 Cypher 在 Neo4j 中是如何工作的：

- **声明式的（Declarative）**：Cypher 是一种声明式查询语言，它不同于那些命令式的语言。我们只需要声明我们要查找的模式，直接告知 Cypher 我们想要什么，而不是告知如何获取它。这一点很重要，因为命令式的查询语言总是假定我们在与数据库交互时，具有如下特质：
  - 操作者是一个程序员，知道如何处理数据库的过程逻辑，并需要以程序的方式构造出来。
  - 熟悉要查询的数据库的结构、大小和语法，能以最优的方式来组织命令查询结果。
  - 这两个假设看起来相当牵强附会。鉴于这一点，很多数据库选择了声明式方法来查询它们的数据。**结构化查询语句（SQL）**就是一个著名的例子。
- **具有表现力的（Expressive）**：很多人注意到 Cypher 语法有点像 ASCII 的风格，它确实可能是。它使用圆括号、方括号以及连接各个部分的箭头这种模式，很容易让人理解查询语言所表达的真实含义。这种**易读的语法设计**背后原因是：**大多数代码被读的次数比被写的次数要多**。因此，富于表现力对于工作在依赖 Neo4j 图数据库管理系统的团队来说是非常有趣的特性。
- **模式匹配的（Pattern Matching）**：Cypher 是一个模式匹配的查询语言。这可能是这门语言更具可视效果的一个方面，它使得人们能够快速容易地构建不同实体之间

的复杂关系。它让我们的大脑很容易理解、掌握。很多人在开始的时候都感觉Cypher是一门非常容易的查询语言。

- **幂等的（Idempotent）**：当Neo公司开始设计Cypher时，查询语言的首席架构Andres Taylor正在使用Scala。Scala是一种基于Java虚拟机的函数式编程语言。函数式编程语言环境的一个关键的特性就是它们是幂等的。通俗地说，这意味着在你的算法中操作一个函数（或者在我们的情况是执行一个查询）的时候，对数据的改变值发生在这个函数第一次执行。同一个函数在同样的数据上的多次执行应当是没有影响的。这不仅仅对函数式程序有巨大的好处，在Neo4j数据库中也一样，因为状态的改变将是幂等的。

知道了这一点，我们可以来看看Cypher中一些关键的操作词，并熟悉在现实世界中的这些概念。

Cypher的最后一个关键特点就是有趣。使用Cypher很有意思，忘掉使用SQL的痛苦吧，看看Cypher查询语句中你的模式，再看看SQL。谢谢Taylor先生！

4.3 与数据很好地结合

CRUD缩写代表创建（Create）、读取（Read）、更新（Update）和删除（Delete），这是针对数据的最基本操作。这是数据库语言必须提供的基本功能。它使得我们可以方便地使用数据。让我们来学习并依次讨论它们。

4.3.1 创建数据

很明显，创建数据的关键字为CREATE。正如我们之前看到的有关莎士比亚的例子：

```
CREATE (romeo:Person {name: "Romeo"})-[:LOVES]-
>(juliet:Person{name:"Juliet"})
```

在例子中从左到右，创建了第一个节点romeo，它带有Person标签和一个字符类型的name属性，属性值为Romeo。然后创建了一个叫LOVES的关系连接到第二个节点juliet。

Neo4j中的关系都是有方向的。因此，我们还可以创建第二个LOVES关系，由juliet连接到romeo：

```
CREATE (romeo:Person{name: "Romeo"})-[:LOVES]-
>(juliet:Person{name:"Juliet"})
CREATE (juliet)-[:LOVES]->(romeo) RETURN romeo, juliet;
```

然而，上面这样创建双向关系不是强制的。不是因为爱有时候只是单向的，而是因为有一种方法可以直接测试两个节点之间是否存在关系，而不必指定方向。我们会在接下来专门讲读取数据的段落看到这一点。

注意：第一个小写的remeo和第一个小写的juliet是**查询范围内**的标识符，如果需要，

可用于在查询中引用各自的节点。

在一个简单的单行查询中，标识符是无用的。因此，下面的语法也完全是有效的：

```
CREATE (:Person{name: "Romeo"})-[:LOVES]->(:Person{name:"Juliet"})
```

下面这个也是合法的，尽管是错误的：

```
CREATE (:Person{name: "Romeo"})-[:LOVES]->(:Person{name:"Juliet"})
CREATE (:Person{name:"Juliet"})-[:LOVES]->(:Person{name: "Romeo"})
```

为什么是错误的？尽管 Cypher 很强大，但它的查询解释器并不神通广大。

该查询语句包含了查询解释器无法区分的节点，它会创建四个完全不同的 Person 节点。语句中没有告诉它我们节点的标识符是什么。我们将在 CRUD 段落之后再回过来看它。

创建节点、关系的指令运行结束后，下面的查询将得到很丰富的内容（我们创建的这些节点代表的人实际上都是来自富有家庭的青少年）：

```
CREATE (romeo:Teen:Person{firstName: "Romeo", lastName: "Montague",
age:
13})-[:LOVES{since:"a long time ago",till:"forever",where:
"Verona"}]->(juliet:Teen:Person{name:"Juliet",lastName:
"Capulet",age: 13})
CREATE (romeo)<-[:LOVES]-(juliet)
```

查询所得到的结果很丰富，是因为 romeo 和 juliet 节点上有两个标签（Teen 和 Person）。这些节点有三个属性（firstName，lastName 和 age），第一个 LOVE 关系也具有一些信息（since，till 和 where），第二个关系没有额外信息。正如之前提及的，创建它不是必需的。这里创建时为了展示关系的箭头从右向左也是可以的。

提示：考虑到通用性和一致性，我们建议相同类型的节点使用相同的属性，尽管 Neo4j 并不强制要求。假如我们想按年龄来统计数据库中所有的青少年，但其中一些青少年没有年龄属性会怎么样？它将会使查询变得复杂，感到头疼或者导致错误的结果。因此，请保持一个标签对应一组属性（0 或者 n 个属性。如果为零，它就相当于一个标记），拥有共同标签的所有节点必须包含标签的所有属性。

你是否发现这一点了？juliet 不像 remeo 节点，它有一个 name 属性，但没有 firstName 属性。这是因为在录入时打错字了，这种情况是经常会出现的。

4.3.2　读取数据

如果我们已经尝试运行过上几节所有的查询语句，那么我们应该有一些 Cypher 编写基础了：

```
MATCH (n:Person)
WHERE n.name="Juliet" or n.firstName="Juliet"
RETURN n;
```

该语句中有两个重要的关键词 MATCH 和 RETURN。它查询数据中有 Person 标签的节

点，同时要符合 name 属性上的一个条件判断。

此外，更具代表性的 READ 查询例子如下：

```
MATCH (n:Person)-[:LOVES]-()
WHERE toLower(n.name)="juliet"
RETURN n ORDER BY n.age ASC
SKIP 2 LIMIT 5;
```

这里，我们在条件判断中加了一个转换为小写的方法。在 age 上加了一个排序功能（从最小到最大，没有 age 属性的节点将放在列表最后）。SKIP 和 LIMIT 用于对结果进行分页。

提示：在 MATCH 中，对于空括号()节点的结果，查询将返回与朱丽叶有 LOVES 关系的任何节点，可能是(p:PoisonousDrink)或者(s:Music:Serenade)。

你是否看到这里的 Cypher 在关系上面没有指明方向？这个语法在“创建数据”小节中提到过。

下面以两种带有不同方向关系的 Cypher 语句做个实验。尝试运行下面这两个相似的查询：

```
MATCH (n:Person)-[:LOVES]->()
```

和

```
MATCH (n:Person)<-[:LOVES]-()
```

两者的结果是不同的。

如果我们想返回朱丽叶爱的所有人或者被罗密欧爱的所有人，可以使用如下查询：

```
MATCH (n:Person{name:"Juliet"})-[:LOVES]-(:Person{name:"Romeo"})
RETURN n;
```

前面的 WHERE 条件被放到了节点上，我们可以看到它实际和上面第二个 CREATE 语句是差不多的。

4.3.3 更新数据

更新一个或者多个节点，首先需要先匹配到（MATCH）它们，然后为我们想更新的属性设置新值。比如一年过去了，我们需要更新 Person 节点的年龄：

```
MATCH (n:Person {name:"Juliet"})
WHERE n.age = 13
SET n.age=14
RETURN n;
```

以每年的批处理方式，我们可以使用如下查询来更新所有的 Person 节点：

```
MATCH (n:Person)
SET n.age=n.age+1
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
RETURN n;
```

你会注意到在我们的小数据库中它有多快。它并没有去触及 age 属性。

注意：动态地添加标签，然后又移除它，这是可以的。

为 age 大于等于 12 和小于等于 18 的 Person 添加 Teen 标签：

```
MATCH (n:Person)
WHERE n.age>=12 AND n.age <18
SET n:Teen
RETURN n;
```

你可以将 age 大于 18 的 Person 节点的 Teen 标签移除。

```
MATCH (n:Teen:Person)
WHERE n.age >18
REMOVE n:Teen
RETURN n;
```

移除一个属性也是可以的（使用 REMOVE n.age）。

如果一个节点不存在某个属性，我们想添加它，这时使用 SET：

```
MATCH (n:Person)
WHERE n.age IS NULL
SET n.age = 0
RETURN n;
```

的确，将 age 设置为 0 没什么用处，但这里只是演示目的。

如果想添加多个属性，可以像下面这样做：

```
MATCH (n:Person)
SET n.age = 0, n.smell="teenSpirit"
RETURN n;
```

或者像这样：

```
MATCH (n:Person)
SET n.age = 0
SET n.smell="teenSpirit"
RETURN n;
```

以上就是更新数据。下面让我们来删除它。

4.3.4　删除数据

删除节点的第一个原则就是它不再连接到其他任何节点。如其他操作一样，先匹配再删除。

如果我们想删除 Montague 先生，查询语句如下：

```
MATCH (r:Person{lastName:"Montague"})
DELETE r;
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

然而，它会失败返回错误，因为该节点有一个 LOVES 关系。

因此，我们要先删除这个关系，或者使用 DETACH DELETE：

```
MATCH (r:Person{lastName:"Montague"}
DETACH DELETE r;
```

现在，Juliet 是单独的了。

现在我们已经初步掌握了 Cypher 基础，并将在后续章节学习更多，下面部分作为参考，也是很有用的。

4.4 Cypher 的关键操作词

每个数据库的查询语言都有一些操作词，它们在每个查询的组成中都有重要的含义。知道这些对我们来说很有用，因为我们将使用它们在我们的数据集上来构造特定的查询。Cypher 的关键操作词如表 4-1 所示。

表 4-1 Cypher 的关键操作词

关键词	功能	例子
MATCH	用于描述数据库应该匹配的模式。它可能是最重要的一部分，总是作为结构化查询组件的开头（它只有一个词，比 SELECT 要短）	MATCH (me:Person)-[:KNOWS]->(friend)
WHERE	用于过滤 MATCH 中只满足特定规则的结果集	WHERE me.name = “My Name” AND me.age > 18
RETURN	用于返回结果集。可以返回路径、节点、关系或它们的属性，或者前面提到部分的聚合体。这是另外一个结构化组件，因为所有的读查询和大多数写查询都将返回一些数据	RETURN me.name, collect(friend), count(*) as friends
WITH	用于将结果集从一个查询部分传给下一个。它很像 RETURN，但不是把数据放到结果集，而是传给查询接下来的部分。它可以转换和聚合结果集，也可以用于分离 READ 和 UPDATE 语句	
ORDER BY SKIP LIMIT	用于排序和对结果集进行分页	ORDER BY friends DESC SKIP 10 LIMIT 10
CREATE	用于创建节点、关系及它们的属性	CREATE (p:Person), (p)-[:KNOWS {since: 2010}]-> (me:Person {name:”My Name”})
CREATE UNIQUE	用于创建那些还不存在的图数据结构	
MERGE	使用索引和锁来匹配或者创建。你可以指定不同的操作，模式存在的情况为 MATCH，模式不存在的情况就为 CREATE	MERGE (me:Person {name: “My Name”}) ON MATCH me SET me.accepted = timestamp() ON CREATE me SET me.age = 42

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

SET, REMOVE	用于更新节点的属性、标签和或者关系	SET me.age = 42 SET me:Employee REMOVE me.fist_name REMOVE me:Contrator
DELETE	用于删除节点和关系	MATCH (me) OPTIONAL MATCH (me)-[r]-() DELETE me

有了这些简单的关键词，我们应该能够开始构建我们的第一个 Cypher 查询了。毕竟，它有点像 ASCII 样式，类似于我们在白板上画的结构图，如图 4-1 所示。

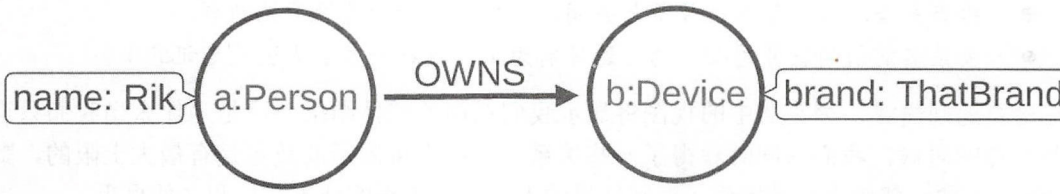


图 4-1　Rick 拥有一个设备

这个用 Cypher 很容易描述，具体如下：

```
(a:Person {name: "Rik"}) - [r:OWNS] -> (b:Device {brand:
"ThatBrand"})
```

为了使它成为一个正确的 Cypher 语句，你只需要将它用 MATCH 和 RETURN 语句连起来即可：

```
MATCH (a:Person {name: "Rik"}) - [r:OWNS] -> (b:Device {brand:
"ThatBrand"}) RETURN a, r, b
```

我们已经看到了一些简单的例子来展示如何开始使用 Cypher。当然，本书的其他地方也能找到一些复杂的例子。

4.5 语法约定

为了使我们与其他 Cypher 用户保持一致，我们强烈建议你遵循如下语法约定：

- 节点别名用小写驼峰（以小写字母开头）。
- 标签用大写驼峰（以大写字母开头）。
- 关系用蛇形大写（类似 IS_A）。
- 属性名用小写驼峰。
- 关键词全部用大写。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

4.6 高级知识

我将会以一件比较重要的事情来开始本节，但不止一件。当我们开始使用 Cypher 时，有些要点需要知道。

出现语法错误后，很多时候可以通过如下方式来改进：

- 检查是否所有的节点括号都是关闭的，缺失右括号是很常见的事。
- 关系名前面的冒号是必需的（到目前为止，我都一直尽力避免忘记它）。

检查返回内容，浏览器中的视图将显示我们查询的整体结果。这是为什么当我们只返回一些节点的时候，我们却同时获得了一些关系。邻接点可展示的数量是有最大上限的，默认为 300。记住：邻接点只是提供了返回结果的上下文，真正的结果在行和文件页里。

查看和修改浏览器的默认设置，点击浏览器左下角的设置图标，可以看到图 4-2 所示的截图。

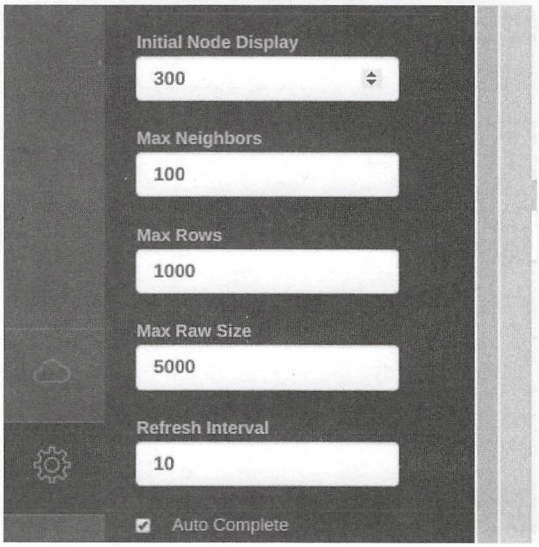


图 4-2 Neo4j 浏览器设置

最后，很重要的一点，不要忘记设置唯一标识符。当然，为了在系统中使用，每个节点都有内在的唯一标识符。

然而，我们一定不要使用它。这不是给我们用的，不要犯这个错误（节点技术上的这个唯一标识符 ID 可能被再分配给新的节点）。让我们回到如何标识一个节点。首先，你必须决定哪个字段（或者哪几个字段）是唯一标识符（类似关系数据库中的主键）。对于有给定标签的所有节点都必须有这个字段（记住：保持一致性）。

语法如下：

```
CREATE CONSTRAINT ON (p:Person) ASSERT p.identifier IS UNIQUE
```

它的作用是创建了一个约束，它会强加于 Person 节点的创建和更新操作上。注意，它只

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

强加于那些 identifier 属性有值的节点。

如果你想移除这个约束，就使用 DROP 删掉它。

```
DROP CONSTRAINT ON (p:Person) ASSERT p.identifier IS UNIQUE
```

在接下来的章节中我们将学习更多有关 Cypher 的知识。

比如数字操作、字符串操作、列表和映射的使用，以及一些更深入的知识。

从现在开始，让我们来点炫一些效果吧！

好友推荐的实例

在图数据库带来的改进中，一个典型应用就是使用在交友和推荐网站（找到一个朋友的朋友或者你的联系人的联系人）。

为了演示，在我们的图数据库中需要更多一点的数据。想象一下有一棵三个 Dude（一个新标签）层级的树。节点没有标识符，因此没有约束，只有一个 name 属性。我们将使用 dudeA1:Dude 来标记第一层的第一个节点，我们的开始节点 leBig:Dude。该节点有三个朋友（在 B 层），分别叫 dudeB1:Dude，dudeB2:Dude 和 dudeB3:Dude，他们在 C 层有四个朋友。

创建这些节点，在浏览器中运行如下 Cypher 语句：

```
CREATE (leBig:Dude {name: "Lebowski"})
CREATE (dudeA1:Dude {name: "A1"})
CREATE (dudeA2:Dude {name: "A2"})
CREATE (leBig)<-[:FRIEND_OF]-(dudeA1)
CREATE (leBig)<-[:FRIEND_OF]-(dudeA2)
CREATE (dudeB1:Dude {name: "B1"})
CREATE (dudeB2:Dude {name: "B2"})
CREATE (dudeB3:Dude {name: "B3"})
```

然后运行：

```
CREATE (dudeA1)<-[:FRIEND_OF]-(dudeB1)
CREATE (dudeA1)<-[:FRIEND_OF]-(dudeB2)
CREATE (dudeA1)<-[:FRIEND_OF]-(dudeB3)
CREATE (dudeA2)<-[:FRIEND_OF]-(dudeB1)
CREATE (dudeA2)<-[:FRIEND_OF]-(dudeB2)
CREATE (dudeA2)<-[:FRIEND_OF]-(dudeB3)
CREATE (dudeC1:Dude {name: "C1"})
CREATE (dudeC2:Dude {name: "C2"})
CREATE (dudeC3:Dude {name: "C3"})
CREATE (dudeC4:Dude {name: "C4"})
CREATE (dudeB1)<-[:FRIEND_OF]-(dudeC1)
CREATE (dudeB1)<-[:FRIEND_OF]-(dudeC2)
CREATE (dudeB1)<-[:FRIEND_OF]-(dudeC3)
CREATE (dudeB1)<-[:FRIEND_OF]-(dudeC4)
CREATE (dudeB2)<-[:FRIEND_OF]-(dudeC1)
CREATE (dudeB2)<-[:FRIEND_OF]-(dudeC2)
CREATE (dudeB2)<-[:FRIEND_OF]-(dudeC3)
CREATE (dudeB2)<-[:FRIEND_OF]-(dudeC4)
CREATE (dudeB3)<-[:FRIEND_OF]-(dudeC1)
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
CREATE (dudeB3)<-[:FRIEND_OF]-(dudeC2)
CREATE (dudeB3)<-[:FRIEND_OF]-(dudeC3)
CREATE (dudeB3)<-[:FRIEND_OF]-(dudeC4)
```

运行语句，你会看到如图 4-3 所示的结果图：

```
MATCH (d:Dude) RETURN d
```

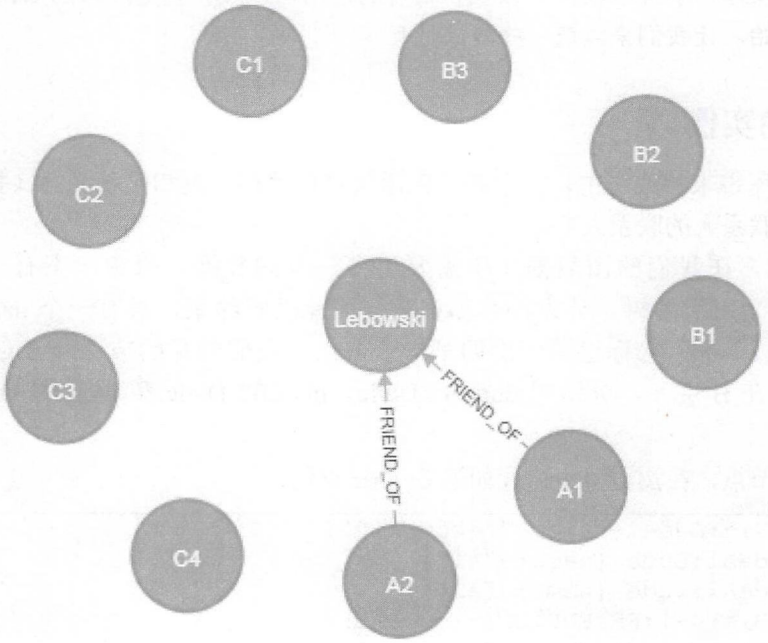


图 4-3　朋友结果图

哦，好多孤立的节点，如此古怪！

双击那些孤立的节点，你是否注意到那些没有名字的灰色节点？是的，但你可能会问为什么呢？因为变量的作用域！

我们用作 ID（如 dudB4）的这些名字是变量名，是为了在一次调用中让查询解析器理解。

我们做了两次调用。我的错，我想向你展示这一点，所以我欺骗了你。是的，我就是那种类型的老师。我认为在学习的时候掉入陷阱比在工作的时候要更好。

我们可能想看看 Neo4j 如何使用前面创建的节点。首先，我们得通过 MATCH 将它们查找出来。

因此，首先我们删除所有的 dude 节点：

```
MATCH (d:Dude) DETACH DELETE d
```

现在，重新执行上面第一个块内的 Cypher 命令。使用下面的这部分命令替代第二块。

```
MATCH (dudeA1:Dude {name: "A1"})
MATCH (dudeA2:Dude {name: "A2"})
MATCH (dudeB1:Dude {name: "B1"})
MATCH (dudeB2:Dude {name: "B2"})
```

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
MATCH (dudeB3:Dude {name: "B3"})
CREATE (dudeA1)<-[:FRIEND_OF]-(dudeB1)
CREATE (dudeA1)<-[:FRIEND_OF]-(dudeB2)
CREATE (dudeA1)<-[:FRIEND_OF]-(dudeB3)
CREATE (dudeA2)<-[:FRIEND_OF]-(dudeB1)
CREATE (dudeA2)<-[:FRIEND_OF]-(dudeB2)
CREATE (dudeA2)<-[:FRIEND_OF]-(dudeB3)
CREATE (dudeC1:Dude {name: "C1"})
CREATE (dudeC2:Dude {name: "C2"})
CREATE (dudeC3:Dude {name: "C3"})
CREATE (dudeC4:Dude {name: "C4"})
CREATE (dudeB1)<-[:FRIEND_OF]-(dudeC1)
CREATE (dudeB1)<-[:FRIEND_OF]-(dudeC2)
CREATE (dudeB1)<-[:FRIEND_OF]-(dudeC3)
CREATE (dudeB1)<-[:FRIEND_OF]-(dudeC4)
CREATE (dudeB2)<-[:FRIEND_OF]-(dudeC1)
CREATE (dudeB2)<-[:FRIEND_OF]-(dudeC2)
CREATE (dudeB2)<-[:FRIEND_OF]-(dudeC3)
CREATE (dudeB2)<-[:FRIEND_OF]-(dudeC4)
CREATE (dudeB3)<-[:FRIEND_OF]-(dudeC1)
CREATE (dudeB3)<-[:FRIEND_OF]-(dudeC2)
CREATE (dudeB3)<-[:FRIEND_OF]-(dudeC3)
CREATE (dudeB3)<-[:FRIEND_OF]-(dudeC4)
```

你看，我们必须先匹配之前我们创建的节点，它们才不会成为孤立的节点。运行语句后你可看到如图 4-4 所示的结果 ：

```
MATCH (d:Dude) RETURN d
```

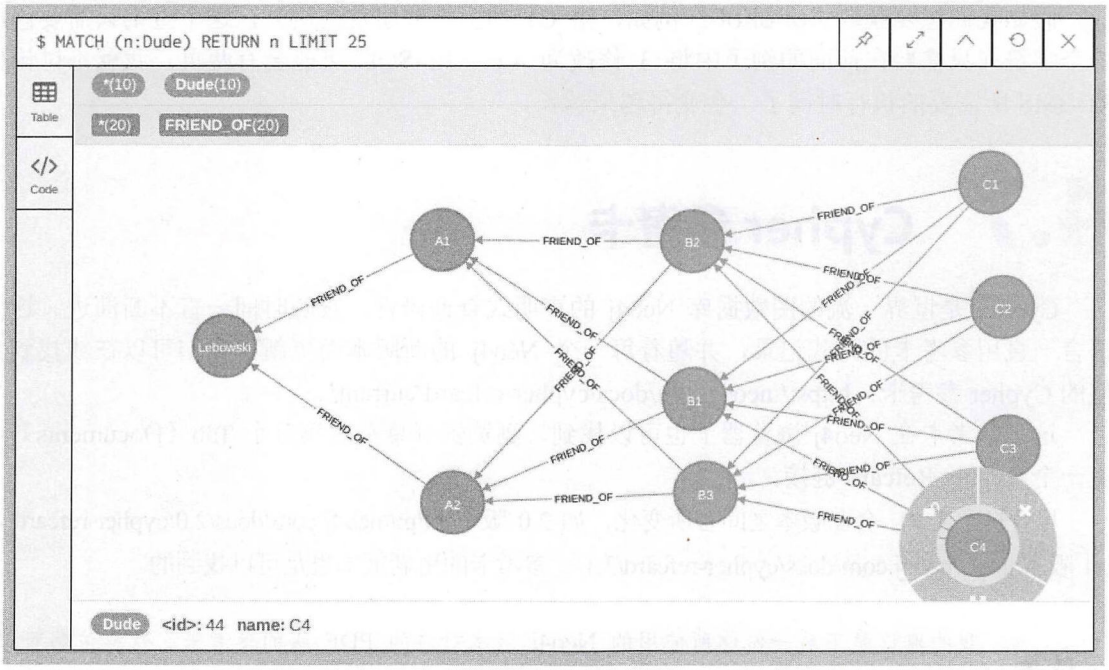


图 4-4　一棵友好的树

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

你不需要双击这些节点来验证他们之间是否有灰色节点。现在我们来看一些查询。

列出 Lebowski 的朋友：

```
MATCH (le:Dude {name:"Lebowski"})<-[:FRIEND_OF]-(some:Dude)
RETURN some
```

这会返回 A1 和 A2。

列出 Lebowski 朋友的朋友，这里有两种方式：

```
MATCH (le:Dude {name:"Lebowski"})<-[:FRIEND_OF]-(someA:Dude)
<-[:FRIEND_OF]-(someB:Dude)
RETURN someB
```

让我们先停一下，想象一张 DUDES 表和 DUDES 表上的一个 DUDE_FRIEND 关联等价的 SQL。这里用到了获取某些人的朋友，以及他们朋友的朋友。

那么，想象一下获取朋友的**朋友的朋友**（FoFoF）的 SQL 语句。我不会将它放到本书因为它将浪费一大段空间和油墨。下面是 Cypher 版的 FoFoF 查询：

```
MATCH (le:Dude {name:"Lebowski"})<-[:FRIEND_OF*3]-(some:Dude)
RETURN DISTINCT some
```

它如期返回 C1 至 C4 节点。

注意两点：第一，我加入了 DISTINCT，假使你以 Table 或者 Text 视图方式查看结果。作为一个练习，尝试加和不加的区别。第二，关系的方向限制了结果，否则 A1 和 A2 节点也会返回，因为他们是 B1、B2 和 B3 的朋友，而 B1、B2、B3 又是 Lebowski 的朋友。FoF 的 F 是 FoFoF，这是我们所要查询的。

假如我们想要获取 FoFoFoF（例如，给 C1 加上一些朋友之后），这个查询只需要修改一个字符（只需要在前面的例子中将 3 修改为 4），用 SQL 可能会有两页。我就不讨论用 RDBMS 所需要的执行时间了，会变得越来越长。

4.7 Cypher 参考卡

Cypher 是世界一流的图数据库 Neo4j 的声明式查询语言，它随时间一直不断演进。这门语言一直用参考卡的方式记录，并随着每一个 Neo4j 的新版本而更新。我们可以在线找到完整的 Cypher 参考卡：https://neo4j.com/docs/cypher-refcard/current/。

这个参考卡在 Neo4j 浏览器上也可以找到。浏览器屏幕左边第三个 Tab（Documents）中有一个 Cypher Refcard 链接。

尽管这个 URL 会在版本之间有所变化，如 2.0 版本 http://neo4j.com/docs/2.0/cypher-refcard/、3.1 版本 http://neo4j.com/docs/cypher-refcard/3.1/，参考卡的更老版本也是可以找到的。

我的建议是下载一份你所使用的 Neo4j 版本对应的 PDF 版的参考卡。这样依然可以在办公室网络出问题的时候查看。在我的智能手机上还有一份，因为它可以帮我解决语法错误。

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

3.2 版本的 PDF 参考卡地址为 https://neo4j.com/docs/pdf/cypher-refcard-3.2.pdf。

4.8 openCypher 工程

自 2015 年 10 月以来，openCypher 工程致力于提供一个开放语法、语言规范、技术兼容的工具和解析器、计划器以及 Cypher 运行时。

Neo4j 网站展示了这样的 openCypher：Neo4j 不仅提供了这个优秀的查询语言，而且主动将其开源，这意味着 Cypher 不会把我们限制在某一个供应商，而是开放给整个图数据库市场，这样对我们非常有好处。

当前，参与到 openCypher 工程的其他供应商包括 Oracle、Apache Spark、Tableau 和 Structr，这些都是来自社区的开源工程。这些工程可能不会提供完整的语言，也可能使用其他的数据库来持久化数据，如 PostgreSQL 或者 Redis。

openCypher 站点位于 http://www.opencypher.org/。

4.9 小结

你做到了！你学到了 Cypher 基础知识。

在数据的 CRUD 部分，你理解了语法、学习了模式、掌握了如何做 CRUD。

现在，你已经是 Neo4j 数据大师一段了，它仅仅是多一些练习。

你有了一些 openCypher 知识，你的 Cypher 技能对于其他数据库也是有用的。希望你能从学习中找到快乐。

现在，让我们进入下一个部分，更精彩的 APOC，即一系列过程的集合。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

第 5 章

过程——APOC

在电影《黑客帝国》中，Apoc 是飞船“尼布甲尼撒”号的工程师兼驾驶员。他最后被叛徒 Cypher 杀害。

以上电影片段只是想勾起《黑客帝国》电影爱好者与 Neo4j 之间的关联。然而这并不能解释什么是 APOC。APOC 在 Neo4j 中代表了以下两个方面:

- 一套完整的组件。
- Cypher 过程。

从 2009 年开始，APOC 作为一个函数和过程的集合，可以在 Cypher 中使用。它包括以下类别:

- 集合操作（排序、最小值、最大值等）。
- 图操作（索引和重构）。
- 文本搜索。
- 转换。
- 排序。
- 地理空间操作。
- 数据集成。
- 报表。
- 获取表示图的元图。

所以，相比《黑客帝国》，Cypher 并没有杀死 APOC，恰恰相反，APOC 提供过程为 Cypher 赋能。

5.1 安装 APOC

第一步是下载与 Neo4j 版本相对应的 APOC 版本（相同的主版本和次要版本号），版本号可以通过打开 Neo4j 浏览器左上方的第一个选项卡获得。以下是根据 Neo4j 版本提供的 APOC 安装方法。

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

5.1.1　机器安装

如果在机器上已经安装了 Neo4j，无论是在 Windows、OS X 还是在 Linux 上，请使用以下命令停止 Neo4j 服务：

```
neo4j stop
```

然后将下载的 jar 文件复制到 plugins 目录下，并重启服务：

```
neo4j start
```

重启后可以直接跳到 5.2 节“检查 APOC 安装”。

5.1.2　Docker 容器安装

首先，需要停止 Docker 容器。

然后，根据第 2 章“Neo4j 基础入门”的介绍，使用 cd 命令进入之前创建的 Neo4j 文件夹并创建一个 plugins 目录:

```
cd ~/neo4j
mkdir plugins
```

在新文件夹中，复制下载的 APOC JAR 归档文件。

然后，我们将重新启动容器，并为这个 plugins 文件夹创建一个新的文件映射:

```
docker run --publish=7474:7474 --publish=7687:7687
--volume=$HOME/neo4j/data:/data
--volume=$HOME/neo4j/logs:/logs
--volume=$HOME/neo4j/plugins:/plugins
```

5.2　检查 APOC 安装

启动服务后，打开 Neo4j 浏览器并输入以下命令查询:

```
CALL dbms.functions() YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN count(name)
UNION
CALL dbms.procedures() YIELD name
WHERE name STARTS WITH 'apoc.'
RETURN count(name)
```

以上的“UNION”查询会返回两行数据（写这篇文章的时候，返回的结果是 98、201）。

注意，我们现在还没开始使用 APOC，这只是一个内置函数。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

UNION 查询会在同一列中返回两个查询结果。

虽然以上查询只返回了 98 个函数和 201 个过程，但是相信不久后，在强大的社区和无可匹敌的 Michael Hunger （@ mesirii)[1]的推动下，很快将达到 365 个函数或过程，我们有一个新的 APOC 计划，就像古老的名言提到：

每天开发一个 APOC，把冗长的语句赶走。

当然，我只是借用了一句名言提出了这个计划。但是我们很清楚，使用 APOC 可以简化查询，因为当我们在代码中有一个 main 主方法并创建了 main one 子方法时，main 方法就会变得简短且易读。

5.3 函数和过程

如前所述，APOC 包含函数和过程。下面我们从**复杂性**上来看看他们有什么不同。

相对于过程，函数相对简单。函数设计的目的是通过读取数据库数据并计算返回一个结果，而过程可以对数据库数据进行修改并返回多个结果。

过程必须使用 CALL 命令来调用，函数则可以直接应用到 Cypher 查询中（见下面加粗部分）：

```
CREATE (p:Person{GUID:apoc.create.UUID()})
```

关于如何使用函数和过程，我们将会在接下来的篇章中详细讲解。

5.4 习惯用法

在介绍 APOC 的主要用法之前，我想说一下我喜欢的一种使用方法。可能你已经注意到了，调用 APOC 看起来很像前面所看到的 UNION 查询的前半部分，确实，CALL 是使用 APOC 过程的关键字。

5.4.1 来自朋友的建议

考虑到 APOC 有约 300 个函数或过程，APOC 提供了一个非常友好的功能。如果要查找名称或描述中包含“meta”单词的所有函数和过程，我们可以使用以下查询：

```
CALL apoc.help('meta')
```

图 5-1 是返回结果。

[1] 译者注：mesirii 是 Michael Hunger 的推特（twitter）账号。

非卖品，仅供非商业用途或交流学习使用 严禁网络传播本PDF，违者责任自负！

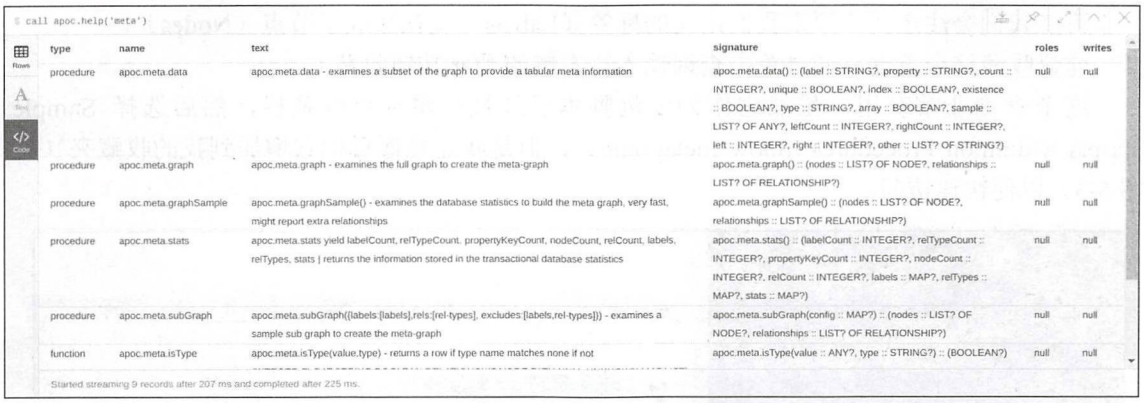


图 5-1　meta 相关的 APOC 搜索结果

图 5-1 中文本（text）一列会提供清楚的文本描述，签名（Signature）列会详细地描述参数和返回值。这个查询非常有用，请放到我们的收藏夹（通过单击查询输入框右边的收藏按钮），方便日后快速访问。

5.4.2　图概述

我们可以用以下查询方便获取图数据库的概述：

```
CALL apoc.meta.graph //  3.1 以下版本
//或
CALL db.schema() // 3.2 及以上版本
```

视图看起来非常简单，如图 5-2 所示。

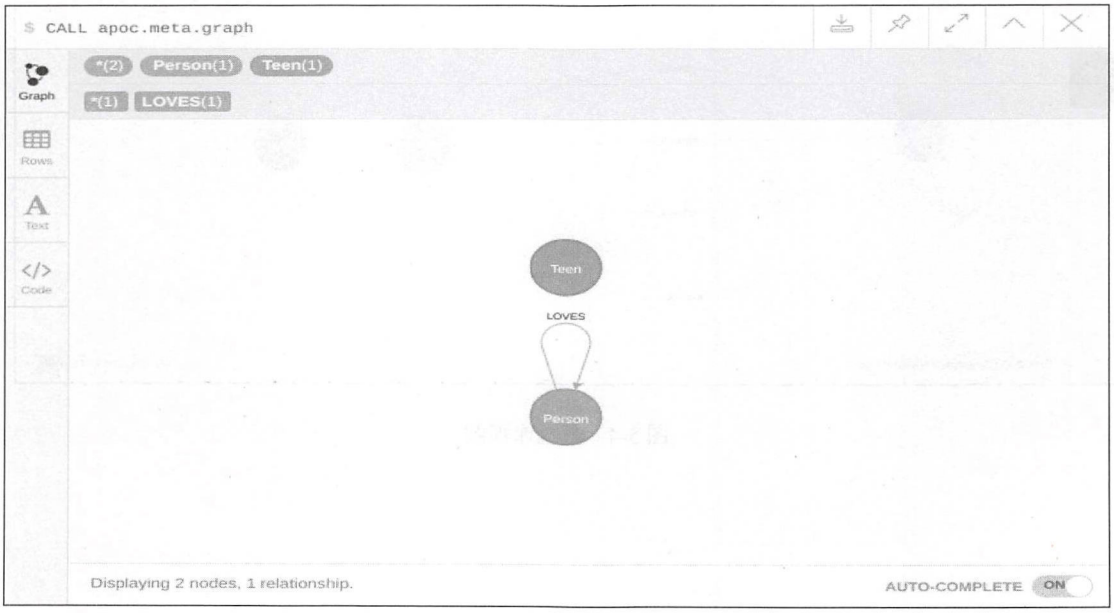


图 5-2　数据库视图

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

可能我们会注意到，原来我们定义的标签（Labels）现在换成了节点（Nodes）。

建议收藏这个查询（通过单击查询输入栏右侧的 Star 按钮收藏）。

这个查询非常有用，虽然它作为示例脚本已经被收藏（点收藏栏，然后选择 Sample Scripts /Common Procedures /Show metagraph），但是我还是愿意把它增加到我的收藏夹（见图 5-3）以便快速访问。

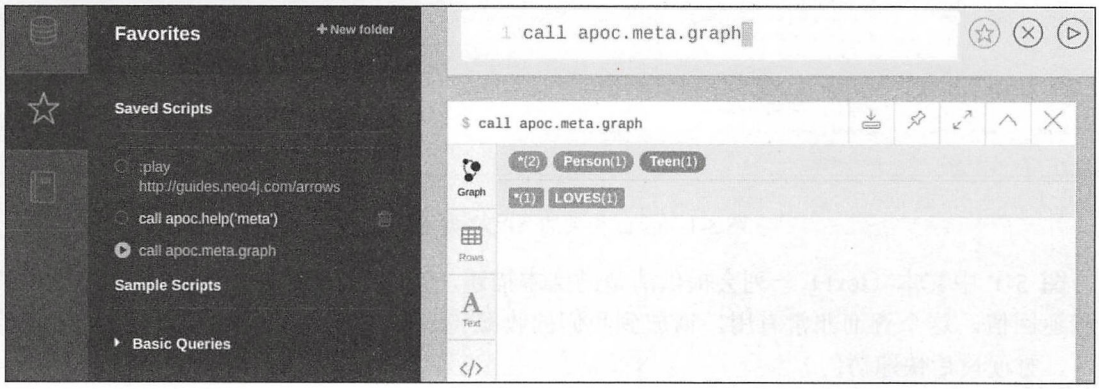


图 5-3　收藏夹

如果需要再次调用收藏的查询，我们只需在屏幕左侧的 Star 选项卡中单击它们即可。如果我们添加了电影数据集，那么概览看起来会更丰富，如图 5-4 所示。

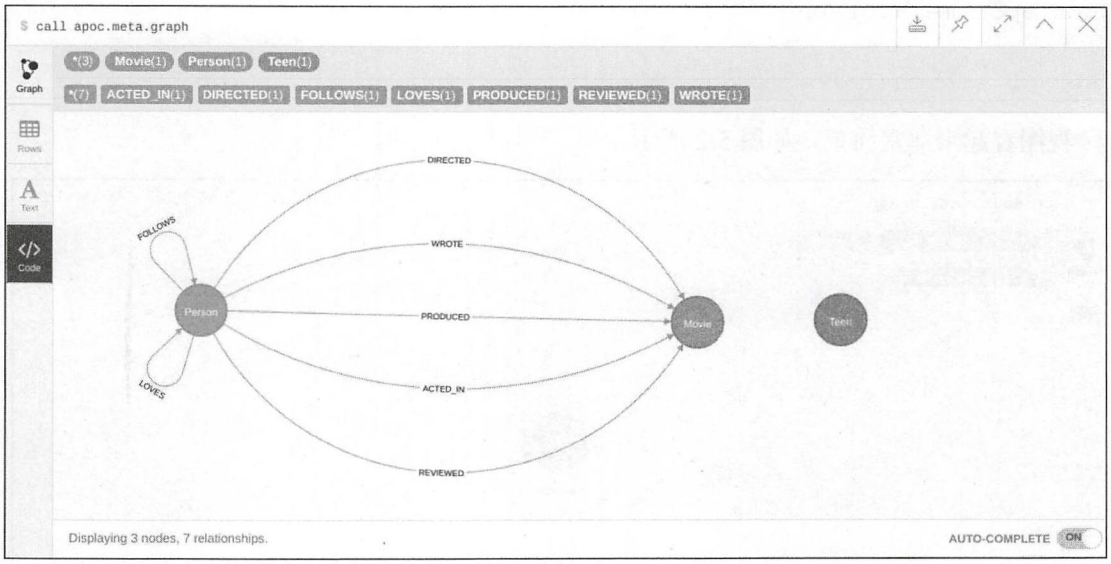


图 5-4　数据库视图

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

5.5 关键用法

下面让我们来看一些关键的使用方法，其他一些 APOC 的用法会在后面的篇章提到，如第 8 章“Neo4j 数据导入”。

5.5.1 安装

首先，我们需要给 APOC 的函数和过程赋权，所以我们在 Neo4j 的配置文件（conf/neo4j.conf）最后增加下面的这行配置，否则的话，我们在执行函数或过程的时候会出现类似这样的错误——apoc.algo.pagerank is not available due to having restricted access rights, check configuration：

```
dbms.security.procedures.unrestricted=apoc.*
```

5.5.2 随机图生成器

严格意义上，下面的过程不能叫随机图生成器，因为算法中可以选择输入参数。虽然这个随机参数意义不大：

```
CALL apoc.generate.ba(1000, 2, 'Person', 'FRIEND_OF')
```

这个例子使用了 **Barabasi-Albert（ba）**模型，会生成上千个节点，并带有 Person 标签和两个 FRIEND_OF 关系。Person 是通用的标签，每个节点都拥有自动添加的名称属性，如图 5-5 所示。

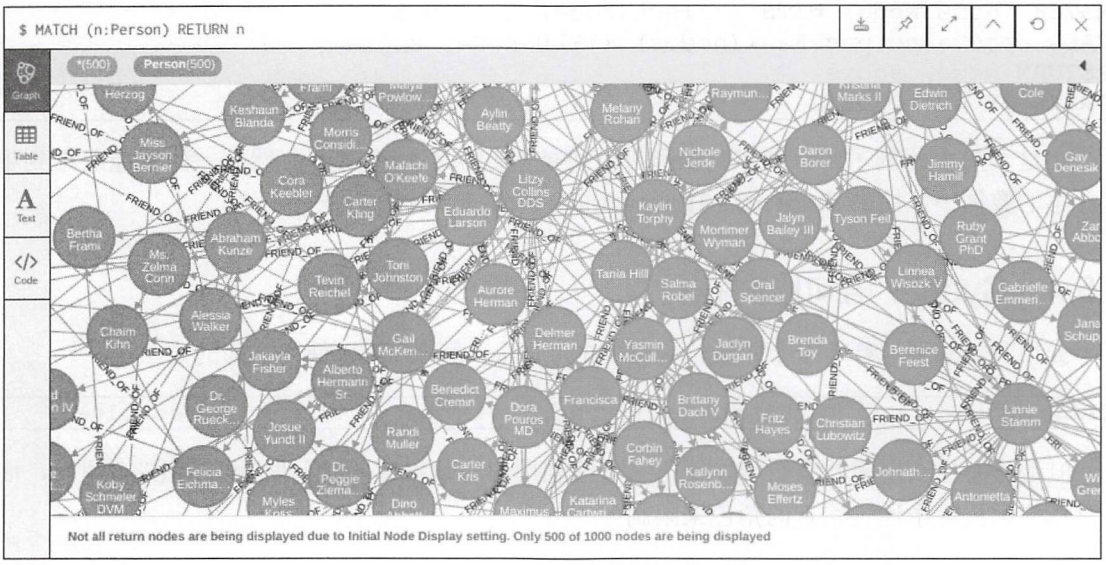


图 5-5　朋友圈

有一个有趣的示例，我们可以创建与自己有关系的节点。如果要找到所有自己的朋友，查询如下：

```
MATCH (p:Person)-[FRIEND_OF]-(p)
RETURN p
```

现在，我们还有其他五种算法可以在 github 上找到，地址为 https://neo4j-contrib.github.io/neo4j-apoc-procedures/index31.html#_generating_graphs。

5.5.3 网站排名

网站排名算法 PageRank 使得 Google 在搜索引擎竞争中脱颖而出，它既计算了关系的数量和质量，又能推断出节点的重要性。规则是：

- 关系越多，节点就越重要。
- 与重要节点的关系越多，节点就越重要。

首先，我们来创建 1000 个节点：

```
FOREACH (id IN range(0,1000) | CREATE (n:Node {id:id}))
```

接着，创建 100 万个关系：

```
MATCH (n1:Node),(n2:Node) WITH n1,n2 LIMIT 1000000 WHERE rand() <
0.1
CREATE (n1)-[:TYPE_1]->(n2)
```

然后，我们调用 pageRank 算法：

```
MATCH (node:Node) WITH collect(node) AS nodes
CALL apoc.algo.pageRank(nodes) YIELD node, score
RETURN node, score
ORDER BY score DESC
```

在这个简单的例子中，节点列表和分数可能看起来不是很有用；然而，如果能应用到更有意思的数据中，比如 Buzzfeed's Trumpworld，你将体会到它的强大之处。

5.5.4 Cypher 语句限时执行

如果我们想要运行一个不超过给定时间的 Cypher 查询，可以使用下面的方法：

```
CALL apoc.cypher.runTimeboxed(CypherStatement, params, timeoutInMs)
```

但是它有什么用呢？我们可以使用它来限制用户对数据库的访问时间（CPU 占用），而不是在应用程序中通过给定的角色来限制：

```
CALL apoc.cypher.runTimeboxed('MATCH (n) return n',NULL, 2)
```

如果数据库中有海量节点，上面的查询在 2 毫秒内如果无法返回全部结果，那么会返回

图 5-6 所示的错误。

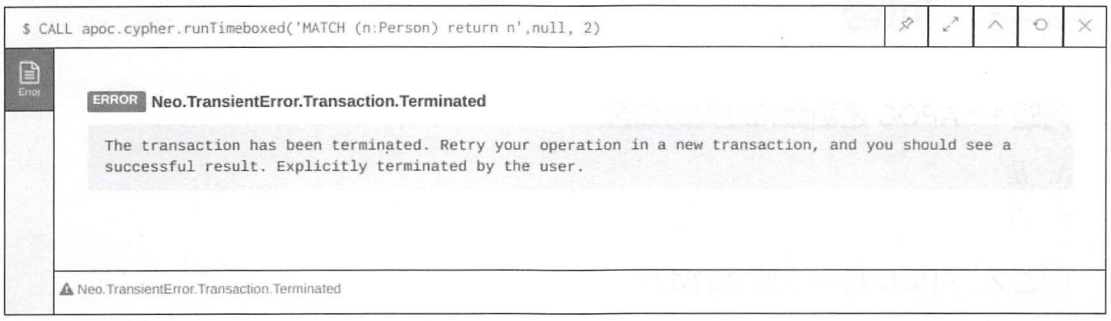


图 5-6　超时，man->terminated

需要注意的是，由于有数据库缓存，相同的查询返回结果在用户之间是相互共享的。如果某个拥有更多时间权限的用户成功执行了查询，在缓存失效前，结果会共享给其他用户。

5.5.5　节点集合连接

连接（Linking）是应用在节点集合中的功能，调用过程可以实现所有节点集合生成同一个关系。

下面的例子是在节点间创建“EARTHBRO_OF”关系：

```
MATCH (p:Person)
WITH collect(p) AS persons
CALL apoc.nodes.link(persons,'EARTHBRO_OF')
RETURN length(persons)
```

如果我们要检查是否只有一个“EARTHBRO_OF”，可以使用下面的 Cypher 查询：

```
MATCH (p:Person)-[r:EARTHBRO_OF]->(m:Person)
RETURN p,count(r)
```

5.5.6　更多 APOC

还有很多 APOC 的内容没有介绍，我们甚至可以专门写一本书。APOC 所涵盖的其他主题还包括:

- **触发器**：给数据库添加动作，例如当为一个节点添加一个标签动作触发时，删除另一个标签。
- **重构**：修改现有的数据模型。
- **自动数据过期**：给你的节点一个时间来实时操作（TTL）。
- 日期操作（字符串转换日期，添加天数等）。
- 与其他数据库的交互（CouchDB、Elasticsearch、MongoDB 等）。

更多内容，请查看 https://neo4j-contrib.github.io/neo4j-apoc-procedures/。

5.6 问答

问题 1：APOC 是函数和过程的集合。

A. 是
B. 否

问题 2：APOC 是一个独立的项目。

A. 是
B. 否

问题 3：在 APOC 中，是否有拨打 911（美国）和 112（欧洲）命令？

A. CALL apoc.help("me Micheal")
B. CALL apoc.112("me, Benoit")

问题 4：APOC 是否可以限制资源使用？

A. 是
B. 否

问题 5：以下哪个命令可以获取图的鸟瞰图？

A. CALL db.schema()
B. CALL apoc.meta.graph()
C. 都可以

问题 6：APOC 由什么组成？

A. 果胶
B. 咖啡
C. 二进制数字
D. 以上都是

问题 7：TTL 是什么？

A. Through The Lens 的首字母缩写，一种用来测量相机光线的方法
B. 节点的生存时间，一个周期后产生的有意义的数据

5.7 小结

在本章中，我们已经了解了什么是 APOC、如何安装和使用 APOC、APOC 由什么构成、函数和过程之间的区别。最后我们还讨论了 APOC 一些主要用法。

通过找到能对 APOC 过程感兴趣的话题，让我们获得了对 APOC 自主学习的能力。现在我们已经知道了 APOC 的好处，可能已经有一些我们正要寻找的函数和过程可以用上了。

现在，让我们进入下一章，学习如何扩展 Cypher 语言。

第 6 章

◂ Cypher扩展 ▸

在熟悉 Cypher 和著名的 APOC 工程之后，接下来将为 Neo4j 实例添加一些函数和过程，这类似于为我们的工程编写专有的 APOC，这些自己编写的扩展将更直接与我们的业务相关。

本章包括以下内容：

- 构建扩展工程（插件）
- 用户自定义函数
- 用户自定义过程
- 定制聚合器
- 非托管扩展
- 流式 JSON 响应

6.1 构建扩展工程

有些工程师总期望能够减轻用户的工作量，因此在 Github 上创建了基于 Maven 的模板工程，以便用户能快速入门这些函数和过程，这样在查找依赖时不会浪费时间。

工程位于 https://github.com/neo4j-examples/neo4j-procedure-template。你可以下载或者克隆一份，然后在我们喜欢的 Java IDE 中打开。在 pom.xml 中，将 neo4j.version 属性的值更新为你使用的 Neo4j 服务的版本。比如我的配置如下：

```
<properties>
  <neo4j.version>3.2.0</neo4j.version>
</properties>
```

用你的 IDE 构建该工程，现阶段这里应该没有错误。在接下来编写的所有代码，我们都将使用同一个工程。

6.1.1　创建函数

这不是一件复杂的事，函数仅仅是 Java 类中的一个方法。这个方法必须添加@UserFunction 和@Description 注解（它们都位于 org.neo4j.procedure 包中）。

现在，我们创建一个函数，如果给定的参数是一个奇数就返回 true，如果是偶数就返回 false。源代码如下：

```
package learningneo4j;

import org.neo4j.procedure.Description;
import org.neo4j.procedure.Name;
import org.neo4j.procedure.UserFunction;


public class OddEven {
    @UserFunction
    @Description("Returns true if the given number is odd")
    public Boolean isOdd(@Name("number")
    Long number) {
        if (number != null) {
            return (number % 2) == 1;
        } else {
            return false;
        }
    }

    @UserFunction
    @Description("Returns true if the given number is even")
    public Boolean isEven(@Name("number")
    Long number) {
        if (number != null) {
            return (number % 2) == 0;
        } else {
            return false;
        }
    }
}
```

我喜欢的部署方式是切换到命令行下，然后输入通常的命令：

```
mvn clean package
```

该命令将在工程的 target 子目录下创建一个 JAR 包，安装函数的过程与安装 APOC 的一样。

复制“procedure-template-1.0.0-SNAPSHOT.jar”JAR 包到 Neo4j 安装目录下的 plugins 子目录。

将这个过程脚本化是一个不错的想法，因为它可能是一个你在以后的开发中会经常重复的过程。

正如我们之前做过的，我们可以通过列出函数来检查它们是否可以工作了。为了测试这个函数，我们将创建一些节点：

```
CREATE (t: Test {val: 15})
```

将 val 的值设置改为奇数和偶数值，多次执行上面的命令。然后，运行如下查询：

```
MATCH (t: Test) WHERE learningneo4j.isOdd(t.val)
RETURN t
```

如我们所看到的，包名是必需的。因此，最好使用一个很棒且比较短的名字，而不是在 Java 中使用经常使用的，如 com.company.project。

APOC 可以帮助我们快速地创建测试数据，尝试运行如下查询：

```
UNWIND apoc.coll.randomItems(range(0,1000),15) AS random
WITH random
CREATE (t:Test{val:random})
RETURN t
```

它将用 1000 以内的随机值创建 15 个节点。

作为练习，可以看一下 apoc.coll.randomItems 的帮助文档。

另外一个函数例子：

```
public class StringHacking {
    @UserFunction
    @Description("Returns the last word of a string")
    //Extending Cypher
    public String getLastWord(@Name("aStr")
    String aStr) {
        if (aStr == null) {
            return null;
        } else {
            int pos = aStr.lastIndexOf(" ");

            if (pos == -1) {
                return aStr;
            } else {
                return aStr.substring(pos + 1);
            }
        }
    }
}
```

现在，我们来看看如何创建过程。

6.1.2 创建过程

过程的原理类似于函数，定制化的代码也需要使用注解。我们的例子很简单，就是保存树（tree）。

过程可以改变图的内容，因此是一种很强大的工具。我们可以想象一个过程，它能够完

成如下任务：

- 从一个互联网源（RSS、JSON 等）获取数据，并将其写入图数据库。
- 清理图数据库的旧数据。

过程能够创建有价值的数据或者为已存在的数据建立关系。在第 4 章“Cypher 入门”中，我们看到了朋友关系的图。我们为每个 last name 创建了节点，并连接 Persons 到它们对应的 LastName 节点。每个有 LastName 标签的节点将有一个 lastName 属性。

我们需要确保每个 lastName 值的唯一性，因此创建唯一性约束：

```
CREATE CONSTRAINT ON (ln:LastName)ASSERT ln.lastName IS UNIQUE
```

让我们确保这里没有 LastName-labeled 节点：

```
MATCH (n:LastName)
```

它应该返回零结果。我们想创建的过程将在一个事务中运行一对 Cypher 查询。它的代码如下：

```
@Procedure
@Description("Regroup all :Person nodes to newly
(re)created :LastName nodes. Recreates all.")
public void regroupByLastName() {
 /* We assume
 CREATE CONSTRAINT ON (ln:LastName) ASSERT ln.lastName IS UNIQUE
 has been run but you can add it to this procedure as an exercise.
 */
 Driver driver = GraphDatabase.driver("bolt://localhost",
AuthTokens.basic("neo4j", "password"));
 try (Session session = driver.session()) {
 try (Transaction tx = session.beginTransaction()) {
 tx.run("MATCH (ln:LastName) DETACH DELETE (ln)");
 tx.run("MATCH (p:Person) " +
 "WITH learningneo4j.getLastWord(p.name) as lw, p " +
"MERGE (ln:LastName {lastName: lw}) " +
"WITH ln,p,lw " +
"CREATE (p)-[:IS_NAMED]->(ln)");
 tx.success();
 }
 }
}
```

这个过程运行的第一个查询为：

```
MATCH (ln:LastName) DETACH DELETE (ln)
```

它删除了有 LastName 标签的所有节点。

然后，它运行了如下代码：

```
MATCH (p:Person)
WITH learningneo4j.getLastWord(p.name) as lw, p
MERGE (ln:LastName {lastName: lw})
WITH ln,p,lw
CREATE (p)-[:IS_NAMED]->(ln)
```

创建带有 LastName 标签的节点，在它和处理的 Person 节点之间建立关系。

这里引入了我们还未见过的 WITH 关键词。WITH 有点像 RETURN，它允许我们将查询分为几个部分。它的作用就是把变量的作用域延续到下一个指令。

下面是前面的查询语句更容易理解的一种写法：

```
MATCH (p:Person)

WITH learningneo4j.getLastWord(p.name) as lw, p
MERGE (ln:LastName {lastName: lw})

WITH ln,p
CREATE (p)-[:IS_NAMED]->(ln)
```

第一个 MATCH 限制查询所有带有 Person 标签的节点。每一个节点都将会被用到接下来的命令中。第一个 WITH 传递变量 p（就像 James Brown 的“传递豌豆”一样），同时声明了一个 lw 变量，它包含了 learningneo4j.getLastWord 函数的结果，函数参数为每个 Person 的 name 属性。MERGE 很像 CREATE，如果节点已经存在它就会失败。最后，在 Person 节点和 LastName 节点之间创建一个 IS_NAMED 关系。

我们可以运行如下查询来查看这个过程的结果：

```
MATCH (a)<-[r:IS_NAMED]-(p:Person)
RETURN a,r,p LIMIT 50
```

现在，如果我们让过程返回这些创建的节点作为结果集，代码应该如下：

```
@Procedure(mode = Mode.WRITE)
public Stream<StringResult> regroupByLastNameVerbose() {
/* We assume
CREATE CONSTRAINT ON (ln:LastName) ASSERT ln.lastName IS UNIQUE
has been run
*/
    Driver driver = GraphDatabase.driver("bolt://localhost",
    AuthTokens.basic("neo4j", "password"));
    try (Session session = driver.session()) {
    try (Transaction tx = session.beginTransaction()) {
            tx.run("MATCH (ln:LastName) DETACH DELETE (ln)");
            StatementResult result = tx.run("MATCH (p:Person) " +
            "WITH learningneo4j.getLastWord(p.name) as lw, p " +
            "MERGE (ln:LastName {lastName: lw}) " +
            "WITH ln,p,lw " +
            "CREATE (p)-[:IS_NAMED]->(ln) "+ " RETURN ln.lastName");
            tx.success();
            return result.list(r -> new
StringResult(r.get(0).toString())
```

```
            ).stream();
        }
    }
}
```

用户自定义的过程，要么返回空，要么返回一个记录列表。这些记录被声明为 public static final 类。

这里，在返回语句中以 Java 8 lambda 表达式的方式使用了 StringResult，它是 APOC 的一部分。lambda 将应用到 Cypher 查询返回的每一个 ln.lastName。第一列被转换成了一个 StringResult 实例。所有的实例作为流返回。

6.2 定制聚合器

在 SQL 领域中一些最常见的聚合器包括 count、sum、min 或者 max。他们一般用 GROUP BY 指令连接。下面是 Cypher 中的一个聚合器例子：

```
MATCH (p:Person)-[IS_NAMED]-(ln:LastName)
RETURN ln.lastName, count(p)
```

自 3.2 版本以后，Neo4j 允许创建你自己的聚合器，这非常简单。我们将演示如何编写一个随机数聚合器，它返回一个随机值。Java 代码如下：

```
package learningneo4j;
import org.neo4j.procedure.*;
public class RandomCount {
    @UserAggregationFunction("learningneo4j.randomCount")
    @Description( "learningneo4j.randomCount - mostly returns a wrong
value " )
    public RandomAggregator randomAggregator() {
        return new RandomAggregator();
    }
    public static class RandomAggregator {
        private long count;
        @UserAggregationUpdate
        public void repeated(@Name("string") Object obj ){ //
parameter given as example
            count +=1;
        }
        @UserAggregationResult
        public Long result(){
            return Long.valueOf( (int)(count * Math.random() + 0.5) );
        }
    }
}
```

下面是使用这个过程的例子。在 Neo4j 浏览器中输入：

```
MATCH (ln:LastName)--(p:Person)
RETURN ln, learningneo4j.randomCount(p.lastName) AS badcount
```

你可以看到我使用了 Cypher 代码中@UserAggregationFunction 的值。正如你从 Java 代码中看到的，我创建了一个新类。除了编码规范外，它的名字不是很重要。重要的是内部类声明为 public 和 static，这是实际上真正起作用的地方。这个主题引入了三个注解：

@UserAggregationFunction
@UserAggregationUpdate
@UserAggregationResult

@UserAggregationFunction 用于注册聚合器使用的名称。你会看到，短的名称会更好一些。注意，当前没有办法列出服务器上的定制聚合器。

6.3 非托管扩展

非托管扩展（Unmanaged Extensions，UE）是一种为 Neo4j 服务器内容服务的方式，但它不被包含在常规的 API 中。严格地说，这是你的代码，运行在服务器上，是为你的数据服务。更准确地讲，这是 JAX-RS 2.0 代码。

JAX-RS 是源于 Java EE 6 中的一个 Java API，它的目标是帮助开发遵循 REST（Representational State Transfer）范例的 Web 服务。对应的 JSR 是#339，它的专家组是由 Java 的领袖级人物构成的，如 Adam Bien 或 Bill Burke。

为了创建一个 Maven 工程，在你的 IDE 初始化创建之后，将如下两个依赖加入到 pom.xml 中：

```
<dependency>
    <groupId>org.neo4j</groupId>
    <artifactId>neo4j</artifactId>
    <version>${neo4j.version}</version><!-- This is a Maven property
-->
    <scope>provided</scope>
    </dependency>
    <dependency>
    <groupId>javax.ws.rs</groupId>
    <artifactId>javax.ws.rs-api</artifactId>
    <version>2.0</version>
    <scope>provided</scope>
</dependency>
```

你可能也会只添加 javax.ws.rs-api 依赖到那些使用了定制函数、过程和聚合器的工程。

6.3.1　HTTP 和 JAX-RS refreshers

在所有的已存在的 HTTP 方法（GET、POST、PUT、DELETE、HEAD）中，我们将集中讲 GET 方法。GET 调用由 HTTP 客户端发起，如浏览器，当用户在地址栏中验证一个 URL 地址时。

最佳实践是，创建类的名字以 Resource 结尾，类所在的包的全名包含 webservices。这个类必须用@Path 注解，并且注解必须有值。当注解中没有参数名时，圆括号内的就是注解的值，例如@Path("thisisthevalue")。这个值将是你写的这个类的服务的一部分。

每个服务也有它自己的@Path 注解和@GET 注解。

因为我们将创建代码来产生 JSON 格式的数据，所以我们必须用@Procedure 来注解方法（MediaType.APPLICATION_JSON）。

下面的例子代码将返回我们使用的数据库中的 last name 列表。

```
package learningneo4j;
import org.neo4j.graphdb.*;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import java.util.ArrayList;
import java.util.List;
@Path("/persons")
public class PersonResource {
    private final GraphDatabaseService database;
    public PersonResource( @javax.ws.rs.core.Context
        GraphDatabaseService database ) {
        this.database = database;
    }
    @GET
    @Path("/lastnames")
    @Produces(MediaType.TEXT_PLAIN)
    public String getLastNames(){
        try (Transaction tx = database.beginTx() ) { // A transaction
is mandatory
            ResourceIterator<Node> nodes
=database.findNodes(Label.label("LastName"));
            List<String> lastNames = new ArrayList<String>();
            Node n = null;
            while (nodes.hasNext()) {
                n = nodes.next();
                lastNames.add((String) n.getProperty("lastName"));
            }
            nodes.close();
            tx.success();
            return lastNames.toString();
        }
    }
}
```

1. 注册

在能够使用新的 Web 服务之前，我们必须注册它的包并把它映射到一个 URL。

打开配置文件 conf/neo4j.conf，然后添加下面这行：

```
dbms.unmanaged_extension_classes=learningneo4j=/learningneo4j
```

现在，我们已经将 learningneo4j 包映射到了/learningneo4j URL 上。

2. 访问

现在，你可以通过如下 URL 访问你的 Web 服务了：

```
http://localhost:7474/learningneo4j/persons/lastnames
```

它的格式为 protocol://server:port，然后加上类中的@Path 注解的值，以及方法上@Path 注解的值。

6.3.2 JSON 响应流式化

HTTP 调用返回的结果会导致在服务器端存储大量的对象，它会使得 JVM 触发比实际情况更多的垃圾回收。为了保持一个较低的内存占用，一种可能就是将结果流式化。

我们将获取一些人的朋友的名字列表，这些人的名字通过一个参数来提供。

用如下代码创建一个类：

```
package learningneo4j;
import org.codehaus.jackson.JsonEncoding;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.map.ObjectMapper;
import java.io.IOException;
import java.io.OutputStream;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.StreamingOutput;
import org.neo4j.graphdb.GraphDatabaseService;
import org.neo4j.graphdb.Label;
import org.neo4j.graphdb.Node;
import org.neo4j.graphdb.Relationship;
import org.neo4j.graphdb.RelationshipType;
import org.neo4j.graphdb.ResourceIterator;
import org.neo4j.graphdb.Transaction;
import static org.neo4j.graphdb.Direction.INCOMING;
import static org.neo4j.graphdb.Direction.OUTGOING;
@Path("/friends")
public class FriendsResource {
   private GraphDatabaseService graphDb;
```

```
    private final ObjectMapper objectMapper;
    private static final RelationshipType FRIEND_OF =
    RelationshipType.withName("FRIEND_OF");
    private static final Label PERSON = Label.label("Person");
    public FriendsResource(@Context GraphDatabaseService graphDb) {
        this.graphDb = graphDb;
        this.objectMapper = new ObjectMapper();
    }
    @GET
    @Path("/{personName}")
    public Response findFriends(@PathParam("personName") final String
    personName) {
        StreamingOutput stream = new StreamingOutput() {
            @Override
            public void write(OutputStream os) throws IOException,
            WebApplicationException {
                JsonGenerator streamingOutput =
                objectMapper.getJsonFactory().createJsonGenerator(os,
                JsonEncoding.UTF8);
                streamingOutput.writeStartObject();
                streamingOutput.writeFieldName("friends");
                streamingOutput.writeStartArray();
                try (Transaction tx = graphDb.beginTx();
                ResourceIterator<Node> persons =
                graphDb.findNodes(PERSON, "name",
                personName)) {
                    while (persons.hasNext()) {
                        Node person = persons.next();
                        for (Relationship relationshipTo :
                        person.getRelationships(FRIEND_OF, OUTGOING)) {
                            Node friend = relationshipTo.getEndNode();
                            streamingOutput.writeString(friend.getProperty
                            ("name").toString());
                        }
                        for (Relationship relationshipFrom :
                        person.getRelationships(FRIEND_OF, INCOMING)) {
                            Node friendComing =
                            relationshipFrom.getStartNode();

streamingOutput.writeString(friendComing.getProperty
                            ("name").toString());
                        }
                    }
                    tx.success();
                }
                streamingOutput.writeEndArray();
                streamingOutput.writeEndObject();
                streamingOutput.flush();
                streamingOutput.close();
            }
        }
        ;
        return
```

```
Response.ok().entity(stream).type(MediaType.APPLICATION_JSON).build()
;
    }
}
```

从 import 中，你可以看到它使用了 jackson 库。因此，我在 pom.xml 中加入了一个新的依赖：

```
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>1.9.7</version>
    <scope>provided</scope>
</dependency>
```

jackson 是一个所有与 JSON 有关的强大的库。根据你读到这些行的时间，这里的版本号可能会有所不同。

重新安装你的插件工程（将它做成了一个脚本，是吗？）并打开你的 Neo4j 浏览器。点击左侧列的 Person 标签，这样下面的查询就会被包含在输入中：

```
MATCH p=()-[r:FRIEND_OF]->() RETURN p LIMIT 25
```

执行它。选一个人，然后读他/她朋友的名字。我选了 Damion Hand。

开一个新的 tab，用这个 URL 浏览你的变量：

```
http://localhost:7474/learningneo4j/friends/Damion%20Hand
```

%20 对应一个空格，因为浏览器对 URL 做了编码。这里使用了一个新注解@PathParam，它与@Path 配合使用。@PathParam 的值是一个变量名（这里为 personName），必须是@Path 值的一部分，这个值被 JAXRS 框架设置到方法的参数变量中。

在我的例子中，personName 变量的值为 Damion Hand。框架已经从浏览器的请求 URL 中解码出来了。

代码主要是在 write 方法中完成的。先用 graphDb.findNodes 获取到给定名字的 Person，然后分别检查节点的外向和内向 FRIEND_OF 关系，并将这些结束节点和开始节点的名字加入 streamOutput 对象。

返回结果的那行代码如下：

```
return
Response.ok().entity(stream).type(MediaType.APPLICATION_JSON).build()
;
```

返回结果是一个基于流的 OK（HTTP code 200）响应，流的类型为代码中所选的。其余部分的代码几乎是 JSON 模板代码了。

6.4 小结

我们在本章学习了更进一步的知识。现在我们能为我们的工程（模块）创建我们自己的函数和过程了。

我们看到了一些能完成小任务的函数，能更新图数据的过程，定制聚合器的构成，非托管扩展是 Neo4j 服务器的定制 REST 入口，以及在我们的项目中付诸实施的新的可能性。

我们来了，我们看见了，我们征服了，或者如凯撒说的：“我来了，我看见了，我征服了”。

```
(you:Reader)-[:LEARNED]->(s:Sentence {language: "latin", meaning: "I
came,I saw, I overcame"}
```

在下一章，我们将看看是否需要提高我们查询的性能。如果是，如何做？准备好！

第 7 章

◂ 查询性能优化 ▸

在本章，我们将学习到如何评估查询语句的性能。那么，如何评价查询语句的好坏呢？这里有一个准则（rule of thumb），就是减少数据库命中（db hit）的次数。

数据库查询命中是图数据库工作的基本单位。我们如何优化它呢？

本章涵盖了如下主题：

- Explain 和 Profile 指令
  - 查询计划
  - 运算符（operators）
- 索引
- 经验法则

7.1 Explain 和 Profile 指令

Explain 和 Profile 是两个 Cypher 指令，它们可以帮助我们看到可能存在的性能问题。Explain 会给出查询的执行计划，但不会执行查询。Profile 将执行查询并返回结果。

执行计划不是由数据库命中数决定，而是由被称为计划器（Planner）的 Cypher 组件定义的。Neo4j 有两种计划器：成本计划器（cost planner）和规则计划器（rule planner）。从 2.2 版本以后，系统默认使用成本计划器。它的功能就是平滑地将查询文本变成运算符（记住这个名字）列表。

7.1.1 查询计划

为了演示，下面给出一个查询计划的例子，它需要从上面往下看。在图 7-1 中可以看到相关的查询命令，这是一个基于 Movies 实例数据集的例子。该数据集可以通过执行“:play movies”命令获得。

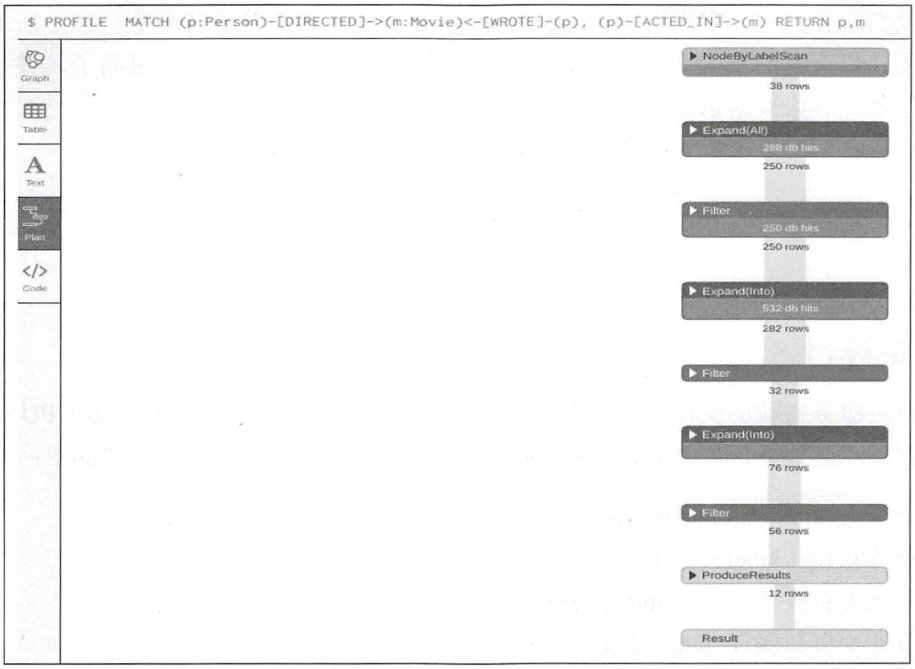


图 7-1　一个短的执行计划

查询计划视图中第一个运算符是 NodeByLabelScan，因为在我们的 Cypher 语句中使用了标签。如果把标签去掉，执行计划将变成图 7-2 所示的样子。

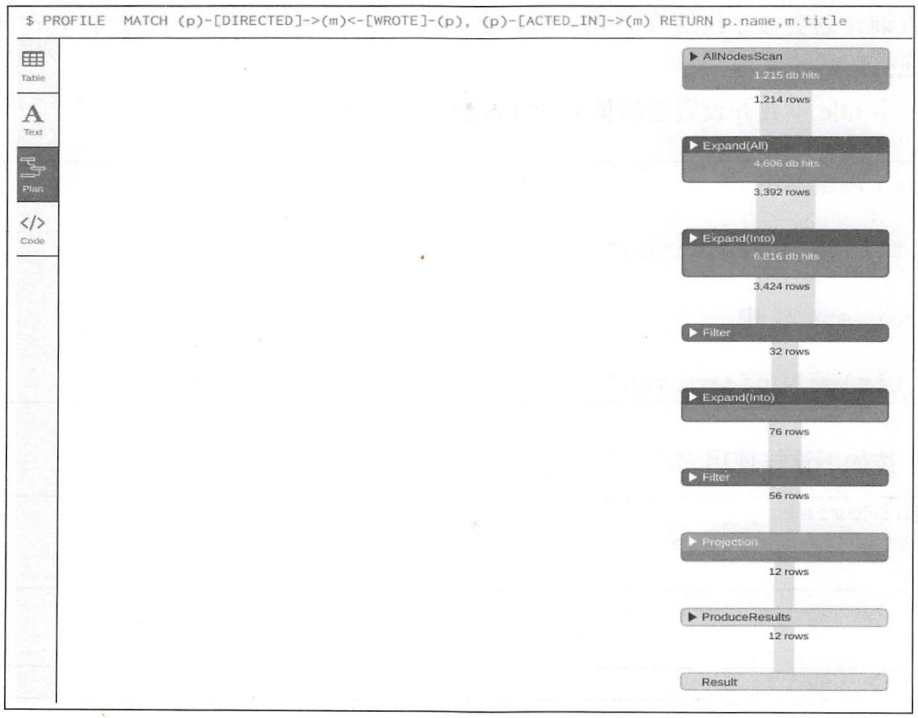


图 7-2　一个长一点的执行计划

正如我们所看到的，它是以 AllNodesScan 运算符开头的。这意味着数据库中所有的节点都会被遍历，以查看它们是否满足查询条件。我们注意到数据库命中数更高了，并且在第一个和第二个节点后连接线也更粗了。

这意味着查询的效率低下，因此我们最好把标签放回去。

当使用 Cypher shell 时，执行计划必须从底部往上看。

7.1.2 运算符

运算符是服务器端的操作。这里有很多种运算符，打个比方，它们有点像 CPU 指令。一些执行的时间比其他长。如 Petra Selmer 和 Mark Needham[1]所述，运算符查找如下：

- 全节点扫描（all nodes scan）
- 标签扫描（label scan）
- 节点索引查找（node index seek）
- 节点索引扫描（node index scan）

很明显，扫描所有节点就是最消耗性能的运算符，剩余部分的性能消耗则没有那么高。面对每个分析过的查询的不同可能性，成本计划器会选择性能消耗最低的那个。

为了达到这个目标，它使用每个查询执行的最新统计。（规则计划器不会使用这些统计。）

因为查询计划会被系统缓存下来，所以最好使用参数而不是常量值，这样查询就不必一次又一次地去解析了。

定义一个 title 参数并设置它的值为 Unforgiven，语法如下：

```
# 在 Neo4j 浏览器中

:param title:"Unforgiven"

#在 Cypher shell 中

:param title:"Unforgiven"
```

然后，按如下这样使用它：

```
MATCH (n:Movie)
WHERE n.title = $title
RETURN n
```

或者：

```
MATCH (n:Movie)
```

[1] 译者注：Petra selmer 是 Neo4j 工程师团队成员，Mark Needham 是 Neo4j 的技术支持工程师。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
WHERE n.title = {title}
RETURN n
```

如果已经导入了电影案例的图数据，就将得到一个结果。

获取已定义的参数列表，使用如下命令：

```
:params
```

当使用 HTTP API 时，如果有很多任务要使用这个查询，你的 Cypher 指令应该写为 MATCH (n:Person {name: {myparam}}) 而不是 MATCH (n:Person {name: 'Dennis'})。

在 Neo4j 浏览器中，你可以粘贴如下查询：

```
:POST /db/data/cypher
{
"query" : "MATCH (n:Movie {title: {title}}) RETURN n",
"params" : {"title" : "Unforgiven"}
}
```

7.2 索引

在 Neo4j 中，索引是用于找到查询的开始点。创建有约束的属性会自动创建索引。或者，我们可以像这样创建一个索引：

```
CREATE INDEX ON :LabelName(propertyName)
```

然而，需要避免为每个标签的每一个属性都创建索引，因为当数据插入的时候服务器需要维护这些索引。我们需要意识到这一点。

获取我们数据库中所有在使用的索引的命令如下：

```
CALL db.indexes
```

7.2.1 强制使用索引

我们可以在查询中强制使用一个索引，如下所示：

```
MATCH (t:Tower {name: ""})
USING INDEX t:Tower(name)
RETURN ...
```

7.2.2 强制使用标签

我们也可以通过使用标签代替使用索引来强制计划器检查某些节点。如果我们查询的节点有两个标签，而且我们知道其中的一个比另外一个更具有限制性，这种情况下就特别有用。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

在这个例子中，我们可能在日本有十亿个位置，为什么我们要查看所有的位置来看它是否有 Tower 标签呢？另外一种做法更好：

```
MATCH (t:Location:Tower)
USING SCAN t:Tower
WHERE t.country='JPN'
RETURN t
```

7.3 经验法则

下面是一些可以遵守的规则，它可帮我们的查询尽可能地达到最好性能。

7.3.1 Explain 所有查询

一旦写完查询语句并且它们的查询结果被验证后，按照本章前面描述的那样，在一个合适的数据集上用 Explain 查看我们所有查询语句的性能瓶颈。

7.3.2 行

在我们的查询计划中，行数从上向下应该是快速递减的。如果不是，那么我们需要考虑使用了足够的标签吗、创建足够多的索引了吗、我们使用的属性上面建索引了吗？

7.3.3 不要过度消耗资源

虚拟的资源也是资源。不要浪费 CPU 周期、内存来返回超过我们所需要的数据。如果我们只想使用节点中的一些属性，就不要将整个节点全部返回。

7.3.4 返回笛卡儿积级别的数据

除非经过授权，在我们的查询中尽量不要做笛卡儿积查询。我们可能已经知道了这点，但有时候它会发生。例如，这是很明显的：

```
MATCH (n), (m)
RETURN n, m
```

注意，返回结果的数量是所有节点数的平方。可以看出，笛卡尔积级别的查询缺乏对不同类型的节点之间关系的描述。

7.3.5 简单

为了让查询更具有可读性，一个避免意外出现笛卡儿积的好方法就是用 WITH 命令把我

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

们的查询切成几块。WITH 允许我们将变量从一块传到另外一块。我们可以传递节点、列表或者值到另外一个语句块。

7.4 小结

我们已经学习了如何检查我们的查询是否在浪费资源，以及如何纠正它们。因为 CPU 周期就是时间和电力，我们的时间是很宝贵的。

我们查看了 Neo4j 的执行计划，并学习如何读懂它们，找出可能的问题所在。我们可以看明白参数是什么，并且它可能会让我们联想起 JDBC 中的 PreparedStatements。我们查看了如何强制使用索引或者标签（我们可能必须在两者之间做个比较，这个依赖于我们的图数据和查询语句）。记住古谚语：让它工作起来，然后，优化它。

现在，让我们切换到一些完全不同的内容：第 8 章 Neo4j 数据导入。另外，导入查询也能调优。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

第 8 章

Neo4j数据导入

没有数据的一个数据库是什么样？那只会是一个空壳。

有多种方法可以将数据导入 Neo4j 服务器。我们前面已经说到过使用 Cypher 语言在富 GUI 的 Neo4j 浏览器客户端实现导入的方法。幸运的是，Neo4j 作为一个大数据类别的数据库，还有其他若干种数据导入的方法：

- 使用 LOAD CSV 导入 CSV 文件
- 导入 JSON 源
- 导入 JDBC 源
- 导入 XML 源

恢复一个备份也算是一种数据导入方法，备份与恢复内容将在附录中单独介绍。

知道如何导入数据，将为我们打开一扇门，一扇包含互联网上各种开放数据的门。我建议使用一些被公开了的商业数据来做导入实验，这些数据更贴近于实际应用。比如我的学生们倾向于使用巴黎咖啡馆的数据集，那里每杯咖啡一欧元，实际上还有更多数据集！让我们开始吧。

8.1 导入 CSV 文件

LOAD CSV 是 Cypher 的内置命令，通过指定一个 CSV 文件，能够使用 Cypher 命令操作这个 CSV 文件的每一行。

CSV 表示 comma-separated value，即逗号分隔值。通常，这种文件的每一列都用逗号分隔，尽管这不是一种很好的分隔方法。更通用的分隔符是分号，这是一种更好的选择，但我个人选择采用 UNIX 的管道符“|”作为分隔符。

文件第一行可能为头部。

CSV 是纯文本，我们不需用电子表格类软件去打开 CSV 文件。

下面是 Neo4j 官网上的一个简单例子。它通过一个 URL 获取数据，且对每一行数据使用 CREATE 命令进行处理。其中加粗显示部分是我们这章将要讨论的：

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
LOAD CSV WITH HEADERS FROM
'https://neo4j.com/docs/cypher-refcard/3.2/csv/artists-with-
headers.csv' AS
line
CREATE (:Artist {name: line.Name, year: toInt(line.Year)})
```

LOAD CSV WITH HEADERS 表示这个数据导入的第一行包含头部（列名）。这样，就可以通过列名来使用相应列。

这个 CSV 文件包含下列很短的内容：

```
"Id","Name","Year"
"1","ABBA","1992"
"2","Roxette","1986"
"3","Europe","1979"
"4","The Cardigans","1992"
```

考虑到我们的数据源中：WITH HEADERS 是可选的（数据源可以不包含头部行，直接以数据开始），就像下列这样也是非常有效的数据源内容：

```
"1","ABBA","1992"
"2","Roxette","1986"
"3","Europe","1979"
"4","The Cardigans","1992"
```

要导入像上面这些内容，相应的 Cypher 语句应该是这样的：

```
LOAD CSV FROM
'https://neo4j.com/docs/cypher-refcard/3.2/csv/artists-with-
headers.csv' AS
line
CREATE (:Artist {name: line[1], year: toInt(line[2])})
```

这时，AS line 表示什么意思呢？

简单地说，它是我们 Cypher 语句中的一种引用，用于指向当前正在处理的数据行。无论数据源中是否有头部，我们都可以在 Cypher 语句中反复使用它们（列）。如果没有，我们只能使用以 0 开始的列索引。使用列索引比列名更容易出错。

我有过这样的导入数据的经历，这些数据是由法国官方发布的公开数据：一些列名有几个词，其余只有一个词。就像表示出生日期的 Birthday 在里面会是这样：Date de naissance，包含 2 个空格。注释字段名也会这样，比如 Description（不考虑 07/2015 转换的形式）。

这些显然会带来麻烦，所以我的经验之谈是尽可能地用反向引号来括住列名：

```
LOAD CSV WITH HEADERS FROM
'https://neo4j.com/docs/cypher-refcard/3.2/csv/artists-with-
headers.csv' AS
line
CREATE (:Artist {name: line.`Name`, year: toInt(line.`Year`)})
```

因为这个例子要重复用，所以选择用短的列名。

无论如何，要清楚下面两点：

- Cypher 语句是可以写在多行的。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

- 我们可以将多个 LOAD CSV 指向同一个数据库源（优先考虑本地文件）。

实际上，我们可以在服务器本地使用这些文件。我们的 Neo4j 服务器安装目录下已经有一个 import 目录。最佳实践做法是加载本地数据时仅从这个目录下导入。首先，考虑可移植性，下面的命令可以在某个管理员下工作但对其他管理员则可能不可移植：

```
LOAD CSV WITH HEADERS FROM
'file:////home/jerome/Tools/neo/neo4jcommunity-
3.2.0/import/level1/artists-with-headers.csv' AS
line
CREATE (:Artist {name: line.Name, year: toInt(line.Year)})
```

因为，我们可以进入$NEO_HOME/conf/neo4j.conf 配置文件并取消这一行的注释：

```
dbms.directories.import=import
```

像这样，指定路径就把 import 作为其根目录（需要服务重启）。比如我们有一个需要导入的文件在$NEO_HOME/import/level1/下，那么 Cypher 语句可以写成这样：

```
LOAD CSV WITH HEADERS FROM
'file:/// level1/artists-with-headers.csv' AS line
CREATE (:Artist {name: line.Name, year: toInt(line.Year)})
```

注意那里是 3 个斜杠！

最后解释一下 toInt(line.Year)。这是一个转换函数，将从数据源里读出的 String 类型数据字段转换成 Integer 类型。

默认状态下，所有数据被当作字符串导入。还有一些其他的类型转换函数，例如：

- toFloat
- toBoolean

我们还可以创建自己的转换过程。

分批导入

如果我们的 CSV 文件又宽（很多列，比如前面提到的 50 列，甚至大于 100 列的）又长（有许多行），导入数据时可能会因 OutOfMemoryException（内存溢出错误）而崩溃。

为避免这种情况发生，一种解决方案是周期性进行事务提交，可以添加下列行：

```
USING PERIODIC COMMIT n
LOAD CSV.....
```

这里，n 的值表示每次事务提交包含的行数。

接下来，我们可以用上面的语句导入一些 JSON 数据试试！

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

8.2 导入 JSON 源

为了实现导入 JSON 源的数据，我们需要使用 APOC（参见第 5 章 过程——APOC）。因此，首先需要确认我们已经安装这些 APOC 了。

我们将要准备一个 JSON 文件，它包含世界各国组成的列表。我们可以在这里查看易读的形式（https://github.com/mledoze/countries/blob/master/countries.json），以及计算机处理的形式（https://raw.githubusercontent.com/mledoze/countries/master/countries.json），我也将其保存的 import 目录里。

为了使用本地的文件副本，需要在 neo4j.conf 里设置下列内容（并重启服务）：

```
apoc.import.file.enabled=true
```

然而，此时我们采用完整路径的方式，且把它作为一个参数，因此，实际上需要将下列例子改成适合我们系统的形式。

先尝试查询看一下这些数据：

```
WITH "file:////home/jerome/Tools/neo/neo4jcommunity-
3.2.0/import/countries.json" AS url
CALL apoc.load.json(url) YIELD value
RETURN value.cca2, value.cca3,value.name.common, value.name.official,
value.capital, value.borders, value.languages;
```

在结果里我们会看到：表示邻国、语言的国家代码（称为 cca2）的集合，比如["BRA","GUF","GUY"] 以及表示国名的{"nld":"Dutch"}，比如 Suriname（苏里南共和国）。也有一个国家多种语言的，比如 Tanzania（坦桑尼亚），语言内容为{"eng":"English","swa":"Swahili"}。

现在，在导入数据之前，让我们解释一下开头的两行：

```
WITH "file:////home/jerome/Tools/neo/neo4jcommunity-
3.2.0/import/countries.json" AS url
```

这个使得我们定义一个名为 url 的变量并可用于下面的 Cypher 命令：

```
CALL apoc.load.json(url) YIELD value
```

这个命令使用已定义的变量（url）作为参数调用 APOC 过程，并且用 YIELD 产生一个名为 value 的变量。

就像我一样，如果大家母语不是英语而且平时不经常使用 yield 这个词，那么请记住它在这种情况下表示产生。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

现在，我们可以做导入了，我们将创建 label（标签）为 Country（国家）的节点。首先，我们先创建一些约束条件（constraint）：

```
CREATE CONSTRAINT ON (c:Country) ASSERT c.cca2 IS UNIQUE
CREATE CONSTRAINT ON (c:Country) ASSERT c.cca3 IS UNIQUE
```

运行下面的语句：

```
WITH "file:////home/jerome/Tools/neo/neo4jcommunity-
3.2.0/import/countries.json" AS url
CALL apoc.load.json(url) YIELD value
CREATE (c:Country)
SET c.cca2=value.cca2, c.cca3=value.cca3, c.name=value.name.common,
c.officialName=value.name.official, c.capitalCity=value.capital ;
```

接下来，我们创建 Language 节点并且连接到 Country 节点上，在相邻的国家间创建关系，这将使图结构越来越复杂。我们需要再创建一个约束条件：

```
CREATE CONSTRAINT ON (l:Language) ASSERT l.code IS UNIQUE
```

为了创建国家节点，我们可以执行下列语句：

```
WITH "file:////home/jerome/Tools/neo/neo4jcommunity-
3.2.0/import/countries.json" AS url
CALL apoc.load.json(url) YIELD value
MERGE (c:Country{cca3:value.cca3})
ON CREATE SET c.cca2=value.cca2, c.cca3=value.cca3,
c.name=value.name.common,c.officialName=value.name.official,
c.capitalCity=value.capital ;
```

MERGE 是一个非常有用的关键词。如果通过 cca3 字段值指定的节点没找到，那么将创建这个节点并且定义在 ON CREATE 之后的 SET 命令将被执行。

如果我们多次运行这个命令，也没有问题。但如果将 MERGE 换成 CREATE，那么第二次执行时将会报错：指示节点已存在。

我们可以从国家数据中导入所有语言，但是我们将等到导入所有的国家节点之后，因为我们将要根据 borders 字段推导创建国家节点间的关系：

```
WITH "file:////home/jerome/Tools/neo/neo4jcommunity-
3.2.0/import/countries.json" AS url
CALL apoc.load.json(url) YIELD value
MERGE (c:Country {cca3: value.cca3})
ON CREATE SET c.cca2=value.cca2, c.cca3=value.cca3,
c.capitalCity=value.capital
WITH c, value, keys(value.languages) AS ki
FOREACH(k IN ki |
MERGE (l:Language{code: k} ) ON CREATE SET l.name =
value.languages[k]
MERGE (c)-[:HAS_LANGUAGE]->(l)
)
```

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

这个例子稍微有些复杂，但也是一个学习更多 Cypher 的机会。它以创建（或合并）相同的国家节点操作开始，然后创建语言节点。

由于读出的 Language 列表不是一个真正意义的列表集合，而是 Map（键值对）的集合，形如{"eng":"English","swa":"Swahili"}，我们需要分离出“键”的集合（这里是 eng 和 swa），并把它运用于指定属性（code，也就是我们创建的唯一性约束条件的唯一标识）。这样用 keys(value.languages) AS ki，创建了一个新的变量 ki，包含“键”（Map——value.languages 的键）的列表。

因此，我们遍历这个列表并且合并语言节点，通过访问 Map 设置其名称。

注意 FOREACH 命令的结构：先是变量声明（k IN ki），然后是 UNIX 的管道符，再就是需要执行的命令，以及右圆括号。

最后，在国家和语言节点之间合并（merge）关系。

为了列出不同的语言及它们在多少国家被官方使用，采用下列查询：

```
MATCH (c:Country)--(l:Language)
RETURN l.code, count(c.cca3) AS count ORDER BY count DESC, l.code ASC
```

现在所有的国家都在库里了，这样就可以根据 JSON 源里的“borders”列表创建他们之间的关系了。还记得 Suriname（苏里南共和国）邻国数据吗：["BRA","GUF","GUY"]，它看上去是 cca3 代码组成的列表，而这个正是在国家节点里作为属性出现的。因此我们就知道如何使用这个列表了，我鼓励大家使用这个代码去创建不同的 HAS_BORDER_WITH 关系。

如果我们第一次这样尝试没有成功，那么可以使用下列命令删除所有的国家、语言节点及它们之间的关系：

```
MATCH (c:Country)--(l:Language)
DETACH DELETE c,l
```

然后使用之前的命令加载它们。为了检查我们的结果，可以执行下面的查询，它将返回 Suriname（苏里南共和国）的结果：

```
MATCH (c:Country) WHERE c.name STARTS WITH "Suri" RETURN c
```

双击查出的结点进行扩展。然后把鼠标指针移至其他国家节点上面，查看细节信息，注意到其 cca3 代码是正确的。下面是一个解决方案：

```
WITH "file:////home/jerome/Tools/neo/neo4jcommunity-
3.2.0/import/countries.json" AS url
CALL apoc.load.json(url) YIELD value
MATCH(c:Country {cca3:value.cca3})
WITH c, value
FOREACH( bc IN value.borders |
MERGE (c)-[:HAS_BORDER_WITH]->(co:Country {cca3: bc})
)
```

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

如果我们对新技能的使用足够自信，可以在这个图数据库上添加更多内容，我们可以采用 https://github.com/opendatajson/factbook.json 网址中的数据。它的内容非常丰富，也就是著名的中情局实录（CIA factbook）。

8.3 导入 JDBC 源

采用 Apoc，通过一个 Java 驱动程序，也可以从关系型数据库中导入数据。因此要做的第一件事就是下载一个适配我们所用源数据库的 JDBC 驱动程序，并将其放入 Neo4j 安装位置的 plugins 目录下。在下面的例子里，我们使用 MySQL 数据库。让我们先安装它。

8.3.1 测试安装

假使我们不愿直接安装 MySQL，可以使用 docker 技术，运行下列命令启动 MySQL 并使 mysql 服务在 3306 端口监听，在此我们使用无密码的 root 用户：

```
docker run -e MYSQL_ALLOW_EMPTY_PASSWORD=yes -p 3306:3306 mysql
```

可以使用我们喜欢的客户端（这里使用 Squirrel）来连接数据库，并输入如下命令去创建一些我们接下来也会导入的数据：

```
CREATE DATABASE learning
use learning;
CREATE TABLE operating_systems (id long not null, name VARCHAR(50),
family VARCHAR(50));
INSERT INTO operating_systems VALUES (1, "DEBIAN", "UNIX");
INSERT INTO operating_systems VALUES (2, "UBUNTU LINUX", "UNIX");
INSERT INTO operating_systems VALUES (3, "LINUX MINT", "UNIX");
INSERT INTO operating_systems VALUES (4, "ANDROID", "UNIX");
INSERT INTO operating_systems VALUES (5, "KALI LINUX", "UNIX");
```

8.3.2 导入所有系统数据

总的来说，这种方法类似导入 JSON 数据源。努力实现一致性是顶级专业人士需要考虑的问题。

那么现在，我们在同一台服务器上就运行了名为 learning 的 MySQL 数据库，使用默认端口，我们想要导入操作系统数据，形成标签为 Os、OsFamily 的节点以及前者到后者的 FROM 关系。ID 列不能重复。也就意味着我们要创建一对约束条件：

```
CREATE CONSTRAINT ON (o:Os) ASSERT o.name IS UNIQUE
CREATE CONSTRAINT ON (of:OsFamily) ASSERT of.name IS UNIQUE
```

创建这些约束条件不是强制性的，除非我们多次运行下列导入命令，因为那样将会导致

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

创建多个同名的节点。我的目标是保持良好、严谨的习惯。

为了访问数据库，JDBC 驱动程序必须载入内存中。用如下指令实现：

```
CALL apoc.load.driver("com.mysql.jdbc.Driver");
```

导入操作系统数据的 Cypher 代码如下：

```
WITH "jdbc:mysql://localhost:3306/learning?user=root" as url
CALL apoc.load.jdbc(url,"operating_systems") YIELD row AS line
MERGE (o:Os {name: line.name})
MERGE (of:OsFamily {name: line.familly})
MERGE (o)-[:FROM]->(of)
```

我们注意到，这个类似导入 JSON 数据。然而，我们可能想知道 YIELD row as line 表示什么。它表示 apoc.load.jdbc 过程返回一个结果 row，且重命名为 line，以用于我们下面的查询中。

下列查询用于验证导入的结果：

```
MATCH (o:Os)-[:FROM]->(of:OsFamily)
RETURN o,of
```

接下来，我们开始导入一些 XML 数据。

8.4 导入 XML 源

如果我们要从 XML 源导入数据，那么 APOC 将提供支持。有效的 XML 源数据有一个主要优点——它已经被一个 XSD 架构（或 DTD）验证过了。

让我们再来导入一些国家数据，但这次是放在洲里面。接下来的 XML 是随手写的较短的，所以若在里面没有发现自己的国家也不要感觉不快。

```
<?xml version="1.0"?>
<earth>
<continent name="Africa">
<country name="Morocco" money="Dinar"/>
</continent>
<continent name="Europe">
<country name="France" money="Euro"/>
<country name="Germany" money="Euro"/>
</continent>
<continent name="Asia">
<country name="China" money="Yuan"/>
<country name="Japan" money="Yen"/>
</continent>
</earth>
```

有些国家属于同一个大洲，有些国家共享一种货币。通常来说，导入之前要先创建约束条件：

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

```
CREATE CONSTRAINT ON (c:Continent) ASSERT c.name IS UNIQUE
CREATE CONSTRAINT ON (mo:Money) ASSERT mo.name IS UNIQUE
```

导入 XML 的原则是创建一个多重 Map，一个在内存中的 Map 的 Map（也就是说那些 Map 可能还包含其他 Map，以此类推），然后运用 Cypher 语句去处理这些 Map。

Cypher 查询如下：

```
WITH 'file:///home/jerome/Tools/neo/neo4j
community-3.2.0/import/countries.xml' as url
CALL apoc.load.xml(url) YIELD value as earth
UNWIND earth._children as continent
MERGE (co:Continent {name: continent.name})
WITH continent,co
UNWIND continent._children as cou
WITH continent, co, cou
MERGE (existingCountry:Country {name: cou.name})
MERGE (money:Money {name: cou.money})
MERGE (existingCountry)-[:INSIDE]->(co)
MERGE (existingCountry)-[:HAS_CURRENCY]->(money)
```

当然，我们根据自己的情况修改导入文件的路径就可以了。不幸的是，不像导入 JSON 时那样，这次需要给出完整的文件路径。

这里出现了一个新的 Cypher 命令：UNWIND，创建一个迭代器（iterator）并使用它。

或许想要问_children 是什么。它是 Map 的一个属性（attribute），在 XML 文件导入时就被创建了。它包含了当前元素（element）的所有子元素。因为，可以想象，每个国家创建时就像下面这样带有子元素而不是简单的属性（attribute）：

```
<country>
<name>France</name>
<money>Euro</money>
</country>
```

这是一个比较烦琐但也相对通用的 XML 书写方法，它将导致在 Cypher 代码里其他层次的复杂性。我们将不得不遍历所有子元素，根据_type 属性（这里，我们叫它货币）的值，使用 CASE 声明（详见 Cypher 参考卡）去做有条件测试（判断）及相应行为，以及通过使用_text 获取相应内容。

作为对读者的善意提醒，我们必须头脑清醒地意识到尽量避免导入烦琐的 XML 数据。

如果我们读到这一章节时有的是自己的数据源，记得检查一下我们查询语句的性能如何。

这样就讲完了关于导入数据的本章内容。

8.5 小结

在这一章里，我们知道了如何从不同类型的数据源导入数据到我们的图数据库，且知道

非卖品，仅供非商业用途或交流学习使用
严禁网络传播本PDF，违者责任自负！

哪一种是优先考虑的。我们知道可以做些有趣的事情，比如导入一欧元咖啡馆的数据集和自由 wifi 凭据的数据集。

严格来讲，我们知道如何导入数据且如何组合它们。我们可以根据自己的兴趣找到这些数据。当我们导入数据并在图数据库中看到它们时，意味着我们真正打开感知这个领域世界的大门。

想一想我们能够组合这些统计信息：犯罪率，贫困情况，酒类销售情况及教育水平。

在下一章，我们将回到地球的话题，来处理地理方面的数据：第 9 章　空间。

严禁网络传播本PDF，违者责任自负！

第 9 章

空　间

一提到“空间”就让人联想到航空航天领域，很抱歉，本章并不能把你训练成一名宇航员。然而，正如你所看到的，使用图数据库也不属于火箭学科，更不能说这远远超出了你的理解力。接下来让我们了解空间方面的知识。

首先，我们必须承认地球的形状接近于一个球体，而不是一个在龟背上游荡的圆盘。让我们开始探索一个全新的领域。

本章将讨论以下主题：

- 回顾地理概念
- 了解 Neo4j 和 APOC 的地理特征
- Neo4j 空间

9.1 空间简介

空间（Spatial）是给节点的位置定位，比如回答，世界上下一位丹尼斯·里奇（Dennis Ritchie，C 语言之父，UNIX 之父）会在哪里？

9.1.1 回顾

地球上的每个位置（或点）都可以通过它的空间坐标——纬度和经度来定位。两者都是以地球为中心而测量出的角度值。赤道上所有点的纬度为零度。正纬度标识赤道以北，负纬度标识赤道以南。

纬度相同的点形成一个虚拟的水平线（圆圈），而经度相同的点也形成一个虚拟的子午线（圆圈）。目前零度子午线确定穿过英国伦敦附近的格林尼治（皇家天文台所在地，不是一个人名杂志）。相传，采用格林尼治取代巴黎作为零度经度，以换取英国使用米制。不幸的是，到目前为止英国还没有使用米制，一定会有人对此合上十指咒骂。

这样在描述空间坐标时有两个标准：纬度-经度（lat-long）和经度-纬度（long-lat）。我们将采用纬度-经度（lat-long）标准。

严禁网络传播本PDF，违者责任自负！

9.1.2 无缺陷塔

我们将在例子中用到三个塔，埃菲尔铁塔和其他两个副本。法国巴黎埃菲尔铁塔的纬度是48.858093、经度是2.294694，它的电梯有时会出故障，使得这古老建筑物成了一个有缺陷的塔。那里有一家厨房在地下的餐馆。

美国内华达州拉斯维加斯城埃菲尔铁塔的纬度为36.112474、经度为-115.1727 37。它也有一个餐厅。

日本东京的东京塔纬度为35.659053、经度为139.745228，和法国的埃菲尔铁塔一样，都是用于发送广播信号。那也有餐厅吗？

9.1.3 什么是空间

Neo4j支持空间，APOC（Awesome Procedures on Cypher，Cypher过程）也支持空间，当然，Neo4j Spatial更具空间性。有必要解释一下。空间能力存在于三块软件中，其空间处理能力从基本级递升到高级。但是，APOC和Neo4j Spatial是Neo4j中两个支持空间特征的插件，变换包名是为了便于过程调用。

9.2 Neo4j的空间特性

从Neo4j 3.0版本开始就内置了有限的空间支持功能。默认支持点和距离。假定点设置了经纬度属性值，就可计算出两点之间的距离。先让我们创建塔，并养成好习惯定义相应的约束：

```
CREATE CONSTRAINT ON (t:Tower) ASSERT t.name IS UNIQUE
CREATE (paris:Tower {name: "Eiffel
Tower",country:"FRA",latitude:48.858093, longitude:2.294694})
CREATE (vegas:Tower {name: "Vegas Eiffel
Tower",country:"USA",latitude:36.112474, longitude:-115.172737})
CREATE (tokyo:Tower {name: "Tokyo
Tower",country:"JPN",latitude:35.659053, longitude:139.745228})
```

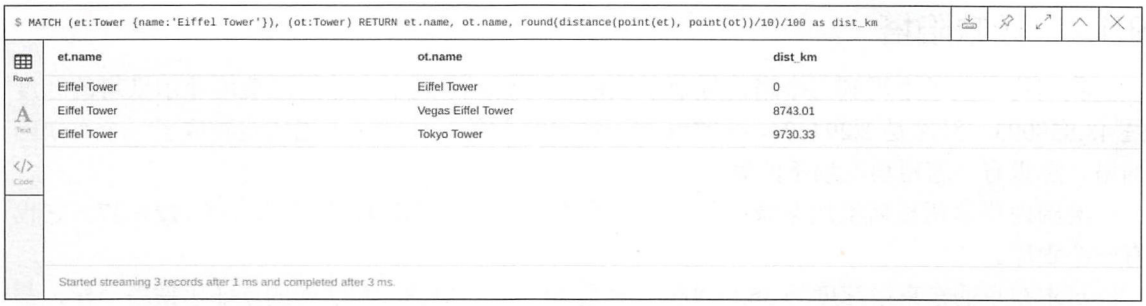
可通过下面查询语句获得埃菲尔铁塔和其他塔之间的距离：

```
MATCH (et:Tower {name:'Eiffel Tower'}), (other:Tower)
RETURN et.name, other.name, round(distance(point(et),
point(other))/10)/100 as dist_km
```

这里的关键是点（point），一个地理位置点是用图数据库中的一个带有纬度和经度属性的节点进行表示，distance函数是通过两点位置计算出来的。

Round函数的作用是舍入到最近的整数，使用小技巧（/10/100），可将数值保留到小数点后两位（见图9-1）。

严禁网络传播本PDF，违者责任自负！



```
$ MATCH (et:Tower {name:'Eiffel Tower'}), (ot:Tower) RETURN et.name, ot.name, round(distance(point(et), point(ot))/10)/100 as dist_km
```

et.name	ot.name	dist_km
Eiffel Tower	Eiffel Tower	0
Eiffel Tower	Vegas Eiffel Tower	8743.01
Eiffel Tower	Tokyo Tower	9730.33

Started streaming 3 records after 1 ms and completed after 3 ms.

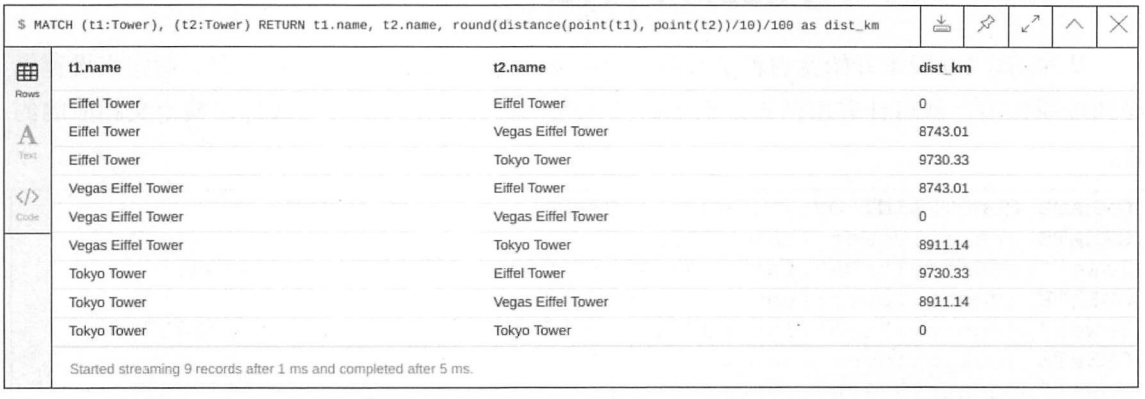
图 9-1　塔之间的距离

距离值是由站点 http://distance.to 提供的，虽然匹配不是很精确。

因为只有三个塔，如果我们想知道东京塔和拉斯维加斯塔之间的距离，一种非常糟糕的查询如同做笛卡儿乘积：

```
MATCH (t1:Tower),(t2:Tower)
RETURN t1.name, t2.name, round(distance(point(t1), point(t2))/10)/100
as dist_km
```

该查询将返回 3 的平方，即 9 行结果。虽然安全，节点数不多不会造成资源的过度使用，但也提供了一个学习如何编写更优查询语句的好机会。任何情况下，请不要做笛卡儿乘积（见图 9-2）。



```
$ MATCH (t1:Tower), (t2:Tower) RETURN t1.name, t2.name, round(distance(point(t1), point(t2))/10)/100 as dist_km
```

t1.name	t2.name	dist_km
Eiffel Tower	Eiffel Tower	0
Eiffel Tower	Vegas Eiffel Tower	8743.01
Eiffel Tower	Tokyo Tower	9730.33
Vegas Eiffel Tower	Eiffel Tower	8743.01
Vegas Eiffel Tower	Vegas Eiffel Tower	0
Vegas Eiffel Tower	Tokyo Tower	8911.14
Tokyo Tower	Eiffel Tower	9730.33
Tokyo Tower	Vegas Eiffel Tower	8911.14
Tokyo Tower	Tokyo Tower	0

Started streaming 9 records after 1 ms and completed after 5 ms.

图 9-2　笛卡儿乘积

下面将用 APOC 提升空间处理的能力。

9.3 APOC 的空间特性

可通过以下方式查询 APOC 的空间过程列表：

```
CALL apoc.help("spatial")
```

APOC 提供了地理编码的可能性，它将地址转换成空间坐标并按距离对路径进行排序。

虽然它只是将地址映射到坐标，但不只是两个浮点数，地理编码并不容易，大量工程师在通过代码进行地理编码的变换。存储汽车行走时拍摄的照片需要消耗大量存储资源，识别图片中的数字需要消耗大量的 CPU 计算资源。

因此，首选是不要自己来处理地理编码，而是依赖一个可被 APOC 透明调用的提供方。

9.4 地理编码

地理编码是将地址转换成地理坐标。两个主要提供商是 OpenStreetMAP（OSM）和谷歌。一次只能使用一个，在更改提供商时需要重新设置并重启服务。

9.4.1 设置 OSM 为提供商

打开常用的 neo4j.conf 文件并添加如下两行：

```
apoc.spatial.geocode.provider=osm
apoc.spatial.geocode.osm.throttle=5000
```

上述定义了提供商并设定间隔时间为 5000 毫秒，以避免拖垮 OSM 服务器，该服务器是免费的，但也需要注意网上行为准则。

9.4.2 设置谷歌为提供商

要使用谷歌作为提供商，须通过 API 控制台获取项目的密钥，请参考网址 https://developers.google.com/maps/documentation/geocoding/get-api-key。当达到每天 2500 次点击的限制后，要收费。

neo4j.conf 文件中的配置如下：

```
apoc.spatial.geocode.google.throttle=10
#查询不至于服务器过载或计费而设置延迟，单位为毫秒
apoc.spatial.geocode.google.key=xxxx
# 谷歌 API 密钥，例如：
AIzaSyCGOqPxaXkODpAKg_teles7tprisqu3icro3yaitpren7dre
```

APOC 支持第三方提供商 OpenCage（https://geocoder.opencagedata.com/）。同谷歌类似，它也是一个付费解决方案，每天免费点击 2500 次。鉴于试用，声称有最宽泛的许可，允许保存查询结果。不要笑，谷歌条款是禁止的。

例如，我想知道我上课的坐标：

```
CALL apoc.spatial.geocodeOnce('143 Avenue de Versailles,75016, PARIS
FRANCE') YIELD location RETURN location.latitude, location.longitude
```

9.5 Neo4j Spatial

Neo4j Spatial（后续简称 NS）如同在“第 5 章 过程——APOC”中介绍的 APOC 一样是一个插件。该插件的主页是 https://github.com/neo4j-contrib/spatial。

同样，需要注意安装的版本要与 Neo4j 版本兼容。请参阅 Github 登录页面上的“发布”标签。发行说明和相关文件是自解释的。（感谢项目负责人 Craig Taverner。）

9.5.1 在线示例

Neo4j 公司员工 William Lyon（https://twitter.com/lyonwj）做了一个在线示例，其网址为 http://www.lyonwj.com/scdemo/index.html。

在左边的工具栏上使用多边形工具来定义形状，在地图上双击可标记餐厅的位置。比如在菲尼克斯城（Phoenix，美国城市）附近查找餐厅，其形状如同蝴蝶（见图 9-3）。

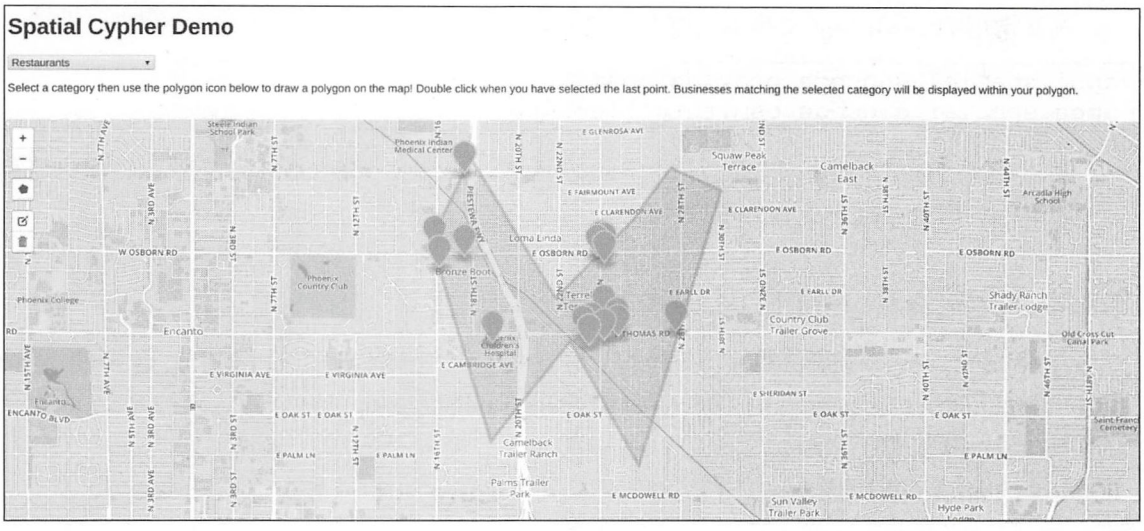


图 9-3 餐馆分布为蝶型

在深入学习 Neo4j 空间知识之前，权当该演示对初学者如同一道餐前点心（开胃菜），接下来将进入 Neo4j Spatial 特性。

9.5.2 Neo4j Spatial 特性

要列出所安装的 Neo4j Spatial 版本所提供的所有特性，可运行查询：

```
call spatial.procedures()
```

其结果返回如图 9-4 所示。

```
$ call spatial.procedures()
```

name	signature
spatial.decodeGeometry	spatial.decodeGeometry(layerName :: STRING?, node :: NODE?) :: (geometry :: ANY?)
spatial.removeLayer	spatial.removeLayer(name :: STRING?) :: VOID
spatial.importShapefile	spatial.importShapefile(uri :: STRING?) :: (count :: INTEGER?)
spatial.intersects	spatial.intersects(layerName :: STRING?, geometry :: ANY?) :: (node :: NODE?)
spatial.updateFromWKT	spatial.updateFromWKT(layerName :: STRING?, geometry :: STRING?, geometryNodeId :: INTEGER?) :: (node :: NODE?)
spatial.getFeatureAttributes	spatial.getFeatureAttributes(name :: STRING?) :: (name :: STRING?)
spatial.bbox	spatial.bbox(layerName :: STRING?, min :: ANY?, max :: ANY?) :: (node :: NODE?)
spatial.addLayer	spatial.addLayer(name :: STRING?, type :: STRING?, encoderConfig :: STRING?) :: (node :: NODE?)
spatial.procedures	spatial.procedures() :: (name :: STRING?, signature :: STRING?)
spatial.withinDistance	spatial.withinDistance(layerName :: STRING?, coordinate :: ANY?, distanceInKm :: FLOAT?) :: (node :: NODE?, distance :: FLOAT?)
spatial.layers	spatial.layers() :: (name :: STRING?, signature :: STRING?)
spatial.importOSMToLayer	spatial.importOSMToLayer(layerName :: STRING?, uri :: STRING?) :: (count :: INTEGER?)
spatial.addPointLayer	spatial.addPointLayer(name :: STRING?) :: (node :: NODE?)
spatial.addWKT	spatial.addWKT(layerName :: STRING?, geometry :: STRING?) :: (node :: NODE?)
spatial.addLayerWithEncoder	spatial.addLayerWithEncoder(name :: STRING?, encoder :: STRING?, encoderConfig :: STRING?) :: (node :: NODE?)
spatial.importOSM	spatial.importOSM(uri :: STRING?) :: (count :: INTEGER?)
spatial.addNode	spatial.addNode(layerName :: STRING?, node :: NODE?) :: (node :: NODE?)
spatial.addPointLayerXY	spatial.addPointLayerXY(name :: STRING?, xProperty :: STRING?, yProperty :: STRING?) :: (node :: NODE?)
spatial.asExternalGeometry	spatial.asExternalGeometry(geometry :: ANY?) :: (geometry :: ANY?)
spatial.setFeatureAttributes	spatial.setFeatureAttributes(name :: STRING?, attributeNames :: LIST? OF STRING?) :: (node :: NODE?)
spatial.closest	spatial.closest(layerName :: STRING?, coordinate :: ANY?, distanceInKm :: FLOAT?) :: (node :: NODE?)
spatial.addWKTs	spatial.addWKTs(layerName :: STRING?, geometry :: LIST? OF STRING?) :: (node :: NODE?)
spatial.layerTypes	spatial.layerTypes() :: (name :: STRING?, signature :: STRING?)
spatial.asGeometry	spatial.asGeometry(geometry :: ANY?) :: (geometry :: ANY?)
spatial.importShapefileToLayer	spatial.importShapefileToLayer(layerName :: STRING?, uri :: STRING?) :: (count :: INTEGER?)
spatial.addNodes	spatial.addNodes(layerName :: STRING?, nodes :: LIST? OF NODE?) :: (count :: INTEGER?)
spatial.layer	spatial.layer(name :: STRING?) :: (node :: NODE?)
spatial.addWKTLayer	spatial.addWKTLayer(name :: STRING?, nodePropertyName :: STRING?) :: (node :: NODE?)
spatial.addPointLayerWithConfig	spatial.addPointLayerWithConfig(name :: STRING?, encoderConfig :: STRING?) :: (node :: NODE?)

图 9-4 空间过程

显然，上面很多过程出现了一些关键词，但还需要进一步解释。即便已经了解 OSM，但还需要认识什么是层（layers）、WKT、shapefiles 和 bbox。

层（layers）可看作是已命名的子集——已导入的节点集的子集，数据导入可以是逐个添加（spatial.addPoint）、列表添加（spatial.addPoints）或文件导入（spatial.importOSMFile）。

添加点时，系统透明地添加到一棵索引树中。随后，该索引树可用于过程调用。

可把层看成树的分枝。与之前介绍的标签所具有的功能相似，能限制节点搜索的数量。层的类型包括 SimplePoint、WKT、WKB 和 OSM。最简单的方法是创建一个 SimplePoint 层，如下所示：

```
call spatial.addPointLayer("Paris")
```

增加 Paris 层，并添加 Eiffel Tower 到该层中：

```
MATCH (etp:Tower {name: "Eiffel Tower"})
CALL spatial.addNode("Paris", etp) YIELD node
RETURN node
```

现在假定给你一个警告：当你删除一个层时，将从数据库中删除所有的节点，并且删除

了与不在该层中节点的关系。下面的代码为演示步骤。

（1）创建一个层：

```
call spatial.addPointLayer("MyWorld")
```

（2）创建 Person 节点至该层：

```
CREATE (me:Person {name:"me", longitude: 1.1, latitude: 2.2})
WITH me
CALL spatial.addNode("MyWorld", me) YIELD node
RETURN node
```

其中，longitude 和 latitude 属性为必填。

（3）创建第二个 Person 节点：

```
CREATE (you:Person {name:"you", longitude: 1.1, latitude: 2.2})
```

（4）创建一个关系：

```
MATCH (you:Person {name:"you"}), (me:Person {name:"me"})
CREATE (you)-[:READ]->(me)
```

（5）检索数据并找到上述两个人和一个关系：

```
MATCH (n:Person) RETURN n LIMIT 25
```

（6）删除该层：

```
CALL spatial.removeLayer("MyWorld")
```

（7）检查数据：

```
MATCH (n:Person) RETURN n LIMIT 25
```

Person 节点消失了，就像拉斯维加斯的魔术表演！

9.6 导入 OpenStreetMap 数据

OpenStreetMap 是一个开放数据社区驱动的倡议，它对全球地理数据（比如街道、道路、站点、公交车站等）进行了重组并共享。

获取数据可以有不同的选择。第一种方法，OSM 站点允许导出屏幕上显示区域的数据。这很适合有精确位置的小文件，比如居住点。在本章后面的示例中，将使用从这个 URL（http://www.openstreetmap.org/export#map=16/48.8576/2.2949）导出的文件。（声明：我并不住在那里。）

第二种方法，OSM 推荐的 GeoFabrik 网站（http://download.geofabrik.de/）允许免费下载更大区域（比如大陆、国家、地区等）的文件。如果需要一个特殊的地理数据集，建议向他们采购。

小文件可以通过 Neo4j 浏览器直接导入，但大文件不建议这样做，比如：一个国家或密集区域。如果缺少 WebSocket 端口将导致导入失败，但导入动画还在继续转动，只有定位到日志尾部

（Linux 下敲击命令：tail -f neo4j.log）才会告知有问题，此时必须杀掉服务器进程。

在这种情况下，运行 neo4j stop 命令将不起作用，因为当没有 WebSocket 端口时，将无法进行连接。这就是为什么用 neo4j console 命令来启动服务器的原因。它不仅在启动服务器后立即登录日志，而且停止服务器只能按下 CTRL+C。

如果想尝试，调用方式如下：

```
CALL spatial.importOSM("/home/jerome/map_ET_Paris.osm")
```

它将创建一个以文件命名的层。也可以用 spatial.importOSMToLayer 过程来指定层的名称。该文件大于 10MB。新节点没有指定标签，所以必须通过属性来了解导入的内容。在 Neo4j 浏览器中，查看左边列，打开第一个选项卡，它看起来像一块超薄煎饼形状，其标题为 Property Keys，其中有一个属性值为 amenity。查看不同的 amenity 值可用下面的语句：

```
MATCH (n) WHERE EXISTS(n.amenity)
RETURN DISTINCT n.amenity AS amenity, count(n) as nbr
ORDER BY nbr DESC, amenity ASC
```

查询结果包含 restaurant（餐厅）、café（咖啡厅）、ATM（自动取款机）、toilets（卫生间）和 bench（长凳）。

如果想通过 amenity 来进行搜索，给该属性添加一个索引看起来非常管用，但无效，因为这些节点没有标签，索引只能创建在标签属性上：

```
CREATE INDEX ON :Label(property)
```

这样，给所有 amenity 节点新增一个 Osm 标签：

```
MATCH (n) WHERE EXISTS(n.amenity)
SET n:Osm
```

接下来，才可能创建索引：

```
CREATE INDEX ON :Osm(amenity)
```

也可以采用 APOC 方式给 amenity 属性值添加标签：

```
MATCH (n) WHERE EXISTS(n.amenity)
WITH n CALL apoc.create.addLabels(n, [n.amenity]) YIELD node
RETURN count(node)
```

注意：CALL 之后必须有 RETURN。

查看数据会发现，坐标值不是在该节点上，而是通过 TAGS 关系关联在另一个节点上（见图 9-5）。

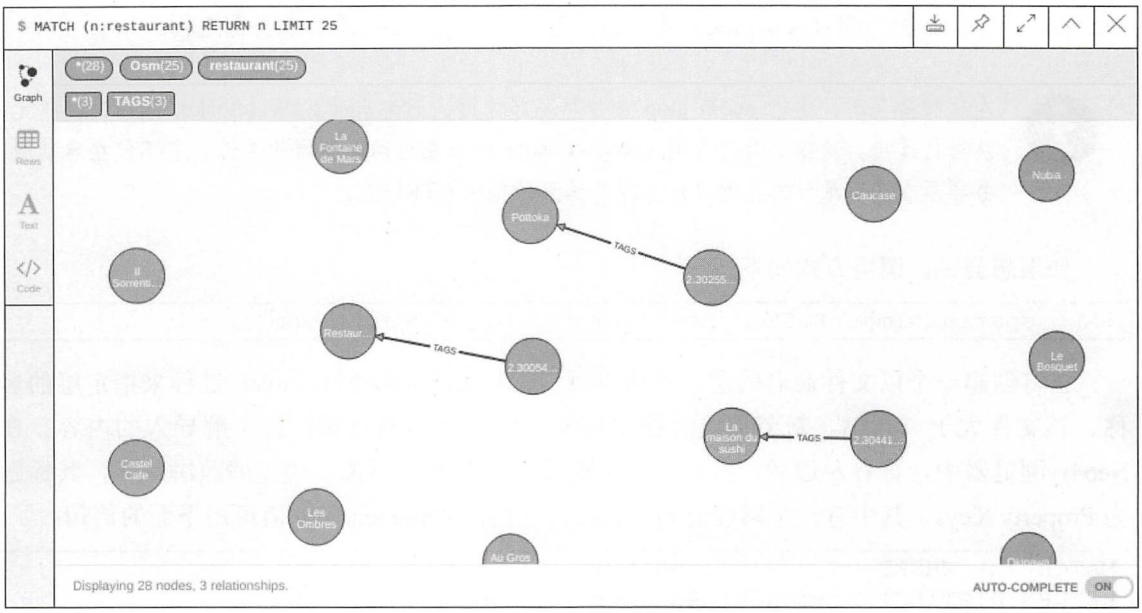


图 9-5　餐馆坐标

如果想列出餐厅及其坐标，查询如下：

```
MATCH (n)-[:TAGS]->(r:restaurant)
RETURN r.name, n.lat, n.lon
```

能发现上面哪里有问题吗？restaurant 标签不是以大写字母 R 开头。需要一个自定义功能（将 amenity 值首字母大写），其结果传递给 addLabels 过程。可以把这当作一个小测验。

9.7 大型 OSM 数据导入

如果深入研究，就会发现有多种数据导入的方法。

9.7.1 简易方法

在处理大型数据集时，不要从 Neo4j 浏览器导入，而应采用 Cypher 命令行。在命令行的 $NEO_HOME/bin 路径下运行 Cypher-shell，接下来进行 Cypher 调用导入过程，如图 9-6 所示。

```
jerome@nitendet ~/Tools/neo/neo4j-community-3.1.6/bin $ cypher-shell
username: neo4j
password: ********
Connected to Neo4j 3.1.6 at bolt://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j> call spatial.importOSM('/home/jerome/OpenSource/spatial/map_ET_Paris.osm'
);
```

图 9-6　Cypher-shell 数据导入

你必须知道，同一个图中不能导入多个 OSM 文件。写这本书的时候 Neo4j 有一个与版本相关的漏洞（bug）。试图导入两个或多个 OSM 文件将引发两个 LegacyIndexProxy 实例不唯一的异常。幸运的是，在阅读本书时已经解决了这个问题。

注意，OSM 文件内容非常丰富，如同项目的更改历史记录，其中大部分数据对你毫无用处。因此，除非 OSM 数据是你业务的核心，请遵循建议：不要在主图中导入 OSM 数据。

在另一台机器上操作副本图，导入感兴趣的区域，然后导出为所需要的 CSV 文件，再将其导入主服务器。下面更直接的导入方式也会有所帮助。

9.7.2 更直接的数据导入方法

另外一种导入 OSM 文件的方法是采用 Maven 和 Java 的命令行。使用下面的命令下载 Neo4j spatial 的源代码，并进行编译：

```
git clone git://github.com/neo4j-contrib/spatial.git
cd spatial
mvn clean install -DskipTests=true
```

接下来，下载所需要的 OSM 文件（获取.bz2 压缩包，解压出里面的.osm 文件），敲入如下命令：

```
mvn dependency:copy-dependencies
java -cp target/classes:target/dependency/*
org.neo4j.gis.spatial.osm.OSMImporter new-osm-db bigBigFile.osm
```

上面将新建一个数据库，全部文件层次都包含在 new-osm-db 文件夹中。这可以在不停止服务器的情况下创建一个全新可用的图。接下来可以一边导入 OSM 数据一边休息了。

9.8 查询卫生间

想象一下，假定一个已知的位置，需要获得最近的卫生间。可使用如下查询过程：

```
CALL spatial.closest('map_ET_Paris.osm', {lat:48.858093,
lon:2.294694}, 1)
YIELD node
MATCH (node)-[:GEOM]-(geo)-[:TAGS]->(t:toilets)
RETURN coalesce(t.name,"unnamed") as name,coalesce(t.fee,'?') as
fee,geo.lat, geo.lon
```

9.9 理解 WKT 和 BBOX

WKT 是前面看到的新关键词列表中的一个，是 well-known-text 首字母的缩写，WKT 是基于文本的标记语言，便于人的理解。其用途是在地图上表示物体，比如点、多边形、线和曲线等，还可以表示 2D、3D 甚至 4D 坐标。

现在你可以立即认出用 WKT 所表达的一个点：

```
POINT(48.858093 2.294694)
```

的确，它就是巴黎埃菲尔铁塔！接下来，没有人能明白下一行所代表的含义：

```
POLYGON( 48.858093 2.294694, 49.858093 2.294696, 49.858093 2.294740)
```

如果它代表什么东西，那它在哪里？（我随机选取的数字。）

你将在其他类型的层中看到带有 WKT 属性的节点。一个 **WKT 层**（WKT layer）是一种可给层指定形状的类型。WKB 是 WKT 的二进制版本。

BBOX 是 bounding box（**边框**）的缩写，一个由两个对角点定义的矩形。在查询中使用边框来返回坐标在该框中的所有节点。

9.10 删除所有地理数据

有可能要从空间树中删除所有的索引节点。可以采用删除所有的层来实现，是一个相当方便的方法。

可尝试使用下面的代码，返回一个空列表：

```
// Removing all spatial layers
CALL spatial.layers() YIELD name AS name
WITH collect(name) AS names
UNWIND names AS nameL
CALL spatial.removeLayer(nameL) // enamel layer :)
CALL spatial.layers() YIELD name AS name2
WITH collect(name2) AS emptyNamesList
RETURN emptyNamesList
```

希望你喜欢这一章，尤其是其中的双关语和俏皮话。

9.11 小结

本章介绍了如何在 Neo4j 中使用地理坐标，无论是采用 APOC 还是使用 Neo4j Spatial 插件。认识了什么是地理编码，以及如何导入 OpenStreetMap 提供的地理数据。因此，只需掌握“第 8 章 Neo4j 数据导入”中的技能，就可以想象一些基于位置的查询，比如周边的餐馆或者在哪里可以找到一辆共享单车。

下一章将学习如何处理数据的安全性。

第 10 章

◂ 安全认证 ▸

在本章中，我们将学习 Neo4j 的安全特性以及如何使用，比如角色和访问限制。安全是我们管理员的首要考虑因素——因为有些数据是比较敏感的，以致需要被限制在指定的用户范围内使用。我们肯定不想让一些来自互联网的某些未知者冒失地删除我们的数据，不是吗？

当用户数达到数百个时，我们需要一个目录（或让更多管理员去创建账户，但不要这样做，这不明智）。

在本章中我们将介绍如下主题：

- 认证与授权
- 角色
- 用户管理
- Neo4j 连接到 LDAP 目录
- 使用 LDAP 配置 Neo4j

10.1 认证与授权

认证是让系统知道我们是谁——我们是属于系统的用户。这通常是通过像登录时使用用户名/密码对的凭据来实现的。

这个用户名/密码对可以由 Neo4j 来管理，也可以由其他系统管理，Neo4j 与之交互验证是否能够进入本系统。指定的其他系统可以是下列技术的目录库：

- LDAP
- 活动目录（active directory）
- Kerberos (与 LDAP 配合使用)

授权是系统对不同用户提供不同级别的访问权限，包含从允许执行不重要的命令，比如更改密码，到允许添加用户或修改图表。在大多数系统中，这是通过角色的使用来实现的。

10.2 角色

Neo4j 的角色有：

- 无
- 只读
- 编辑
- 发布
- 架构
- 管理员

上面是按权限范围的升序来排列的，其中管理员的权限是最大的。当创建一个用户时，最初的角色是无，然后管理员通过角色进行授权。

上述每个角色的权限都按序提升，并且每个角色的权限都包含前面角色的权限。让我们浏览一下每个角色的权限描述。

- **无**：这是默认角色。它允许我们改变自己的密码和查看自己的信息。除此之外没有更多权限，而且这是最基本的角色。
- **只读**：该角色可以读取全图。
- **编辑**：该角色可以读、写全图的已有数据，并且可以用已有的标签和属性创建新的节点，以及用已有的名字创建新关系。
- **发布**：该角色可以创建任何新节点、新标签、新关系。
- **架构**：该角色可控制图的索引及约束（通过 CREATE 和 DROP）。
- **管理员**：该角色可以创建用户，授予角色，及拥有终止某个查询的权利。因此，这个角色是最有威力和拥有最多权利的。“Neo4j”这个账户就被授予管理员角色。（记得要及时修改初始密码哦！）

其他角色

管理员可以使用下列命令创建和删除角色：

```
CALL dbms.security.createRole('newRoleName')
CALL dbms.security.deleteRole('newRoleName')
```

这些角色被用于定制过程中（参见第 6 章 Cypher 扩展），通过使用@Procedure 注解，如下所示：

```
@Procedure(allowed='newRoleName')
```

我们可以关注一下运行下列命令的结果中的 roles 列：

```
CALL dbms.procedures()
```

这个命令适合所有拥有只读角色权限及以上的用户。

10.3 用户管理

管理员可以在 Neo4j 浏览器中用下列命令创建一个新用户，由此出现一个新界面：

```
:server user add
```

在正在创建的新用户的界面上填入用户名、角色和密码。通过下列命令显示用户列表以查看创建的用户是否为正常状态：

```
:server user list
```

在这个出现的列表上，可以对一个用户进行失效或删除操作。

当然，我们也可以不使用带有输入框和按钮的 Web 界面。还有更多的内置过程用于用户管理，通过 Neo4j 浏览器或通过 Cypher shell 外壳访问，如表 10-1 所示。

表 10-1 用于用户管理的内置过程

功能	过程
创建一个用户	CALL dbms.security.createUser(username, password, requirePasswordChange)
删除一个用户	CALL dbms.security.deleteUser(username)
角色分派	CALL dbms.security.addRoleToUser(roleName, username)
角色分离	CALL dbms.security.removeRoleFromUser(roleName, username)
创建角色	CALL dbms.security.createRole('newRoleName')
删除角色	CALL dbms.security.deleteRole('newRoleName')
修改指定用户密码	CALL dbms.security.changeUserPassword(username, newPassword, requirePasswordChange)
自身密码修改	CALL dbms.security.changePassword(password, requirePasswordChange)
列出全部用户	CALL dbms.security.listUsers()

接下来，我们讨论与 LDAP 关联的内容。

10.4 Neo4j 连接到 LDAP 目录

我们将以一条建议引出下面的内容。如果使用集群，那么将每个节点连接到同一个 LDAP 目录的方式是比较可行的，因为用户对每个节点而言都是本地的，是不需要传播的。

那么，我们开始连接吧。我们将使用基于 OpenLdap 镜像的 Docker 容器。

启动目录

我们先准备一个用来存放配置的目录。在 shell 环境里，在我们的 home 目录下执行下列命令：

```
mkdir data/ldap/environment -p
mkdir data/ldap/db -p
```

在第一个新建的目录里，新建一个名为 users.ldif 的文件，包含如下内容：

```
dn: dc=learningneo4j,dc=com
dc: learningneo4j
description: Directory of all the readers
objectClass: top
objectClass: dcObject
objectClass: organization
o: Readers Directory
dn: ou=users,dc=learningneo4j,dc=com
ou: users
objectClass: organizationalRole
cn: users
# ================================================================
ROLES
dn: ou=ldapreaders,dc=learningneo4j,dc=com
objectClass: top
objectClass: organizationalRole
roleOccupant: cn=ureader,ou=users,dc=learningneo4j,dc=com
cn: ldapreaders
dn: ou=ldapproducers,dc=learningneo4j,dc=com
objectClass: top
objectClass: organizationalRole
roleOccupant: cn=uproducer,ou=users,dc=learningneo4j,dc=com
cn: ldapproducers
dn: ou=ldaparchs,dc=learningneo4j,dc=com
objectClass: top
objectClass: organizationalRole
roleOccupant: cn=uarch,ou=users,dc=learningneo4j,dc=com
cn: ldaparchs
dn: ou=ldapadmins,dc=learningneo4j,dc=com
ou: ldapadmins
objectClass: top
objectClass: organizationalRole
roleOccupant: cn=uadmin,ou=users,dc=learningneo4j,dc=com
cn: ldapadmins
# ================================================================
USERS
dn: cn=ureader, ou=users,dc=learningneo4j,dc=com
objectclass: inetOrgPerson
sn: ureader
userpassword: ureader
memberOf: ou=ldapreaders,dc=learningneo4j,dc=com
cn: ureader
#ou=users,
```

```
dn: cn=uproducer,ou=users,dc=learningneo4j,dc=com
objectclass: inetOrgPerson
cn: Producer User
sn: uproducer
userpassword: uproducer
memberOf: ou=ldapproducers,dc=learningneo4j,dc=com
dn: cn=uarch,ou=users,dc=learningneo4j,dc=com
objectclass: inetOrgPerson
cn: Architect User
sn: uarch
userpassword: uarch
memberOf: ou=ldaparchs,dc=learningneo4j,dc=com
dn: cn=uadmin, ou=users,dc=learningneo4j,dc=com
objectclass: inetOrgPerson
cn: Admin User
sn: uadmin
userpassword: uadmin
memberOf: ou=ldapadmins,dc=learningneo4j,dc=com
```

LDIF 代表 LDAP 数据交换格式（LDAP data interchange format）。这是一种易读的文本格式，尽管不是很吸引人，但对加载和更新 LDAP 目录很有用。

接下来我们使用下列命令来启动一个新的容器：

```
docker run --name="ldapcon" --env
LDAP_ORGANISATION="LearningNeo4jWithJerome" --env
LDAP_DOMAIN="learningneo4j.com" --env
LDAP_BIND_DN="cn=admin,dc=learningneo4j, dc=com" --env
LDAP_BASE="dc=learningneo4j, dc=com" --env
LDAP_ADMIN_PASSWORD="agentSmith"
-p389:389 --volume
$HOME/data/ldap/environment:/container/environment/01-
custom --detach osixia/openldap:1.1.9
```

这个表明我们将运行一个域名为“learningneo4j.com”的 LDAP 服务器。我们暴露 LDAP 的默认端口 389，并允许容器访问 users.ldif 文件。现在，为了不想让大家操作太烦琐，我们使用 LDIF 格式文件来创建所在组织内的用户组和用户，命令如下：

```
docker exec ldapcon ldapadd -x -D "cn=admin,dc=learningneo4j, dc=com"
-w
agentSmith -f /container/environment/01-custom/users.ldif -H
ldap://127.0.0.1/ -v
```

这将在我们的容器内使用 users.ldif 文件执行 ldapadd 命令。结果是一棵树的形式，建议大家使用形如 JXplorer 或 Apache Directory Studio 的图形界面查看——前者较轻量级而后者更好一些。进行账户/密码为 admin/agentSmith 的登录，创建一个指定 127.0.0.1 的 389 端口的连接。

如果想在我们的运行容器中执行 shell 命令，可使用如下语法：

docker exec -it ldapcon /bin/bash

接下来是时候让 Neo4j 使用我们的 LDAP 容器了。

10.5 使用 LDAP 配置 Neo4j

大家应该已经知道这个配置工作应在$NEO_HOME/conf/neo4j.conf 中进行。一个必选项是 LDAP 组到 Neo4j 默认角色的映射。将下列内容填加到文件结尾：

```
dbms.security.auth_provider=ldap
dbms.security.ldap.host=ldap://127.0.0.1:389
dbms.security.ldap.use_starttls=false
dbms.security.ldap.authentication.mechanism=simple
dbms.security.ldap.authentication.user_dn_template=cn={0},ou=users,dc
=learn
ingneo4j,dc=com
dbms.security.ldap.authentication.cache_enabled=false
dbms.security.ldap.authorization.use_system_account=true
dbms.security.ldap.authorization.system_username=cn=admin,dc=learning
neo4j,
dc=com
dbms.security.ldap.authorization.system_password=agentSmith
dbms.security.ldap.authorization.user_search_base=dc=learningneo4j,dc
=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)
(cn={0
}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf
dbms.security.ldap.authorization.group_to_role_mapping=\
"ou=ldapreaders,dc=learningneo4j,dc=com"= reader; \
"ou=ldapproducers,dc=learningneo4j,dc=com"= publisher; \
"ou=ldaparchs,dc=learningneo4j,dc=com"=architect; \
"ou=ldapadmins,dc=learningneo4j,dc=com"=admin
```

接下来重新启动 Neo4j 服务，然后我们将可以使用定义在 users.ldif 中的用户凭证进行登录，形如 ureader/ureader 或 uadmin/uadmin 这样的用户/密码对。

如果我们想切换回之前的认证系统，只需简单改变下列配置：

```
dbms.security.auth_provider=ldap
```

替换成如下行：

```
dbms.security.auth_provider=native
```

10.6 问答

问题 1：角色是干什么的？

A. 人员分组用于授权
B. 人员分组用于认证
C. 舞台表演

问题 2：Neo4j 中有多少角色？

A. 5 个
B. 4 个
C. 更多

问题 3：哪些角色可以管理用户？

A. 发布
B. 架构
C. 管理员

问题 4：使用 LDAP 目录带来哪些好处

A. 复用已有的硬件、软件和过程
B. Neo4j 管理员不用在 Neo4j 中不得不创建所有用户
C. 包含上面两点
D. 无

10.7 小结

在本章针对企业版 Neo4j 的内容中，我们学习了角色的使用及其重要性、如何创建用户、如何创建角色以及如何针对过程调用授予特定的角色——这对于限制用户图的展示是很棒的。能否访问我们称之为 Neo4j 浏览器的 Web 应用程序这种限制的设定将贯穿于整本书（在 Neo4j.conf 中配置这个）。

我们已经学习了如何用选定的数据集运行一个容器化的 OpenLDAP 服务器。将此目录连接到 Neo4j 服务器，以便将身份认证委托给 OpenLDAP 服务器。这样，Neo4j 管理员不必在 Neo4j 中重新创建用户，用户也没有必要用另一个密码去记住。

在接下的“第 11 章 Neo4j 可视化”中，我们将看到完全不同的领域和学习如何可视化我们的数据。

第 11 章

◂Neo4j 可视化▸

在本章中，我们将探讨 Neo4j 图数据库管理系统的可视化领域，包括 Neo4j 可视化的重要性和实现可视化的技术工具，在最后，我们将讨论这个领域内的一些常见做法和常见错误。

我们将讨论以下主题：

- 可视化的重要作用（特别是图的可视化）
- Neo4j 图可视化解决方案的基本原理和组件
- 目前流行的可视化组件库和解决方案

下面让我们进入主题。

11.1 可视化的重要作用

在本节中，我们将花时间讨论图可视化之所以如此重要的原因，并重点介绍各种不同的可视化解决方案中的一些基础技术。

11.1.1 为什么图数据可视化很重要

一张图片胜过千言万语。

图数据可视化化之所以如此重要是有很多原因的，我们在前面的内容中也介绍过。特别是在图数据库应用的项目中，图数据的可视化会对整个项目的决策起到重大的作用，下面让我们开始探讨吧。

为什么本章将要讨论的图表可视化以及之前我们已经看到并将在后面展示的可视化非常好？由于数量有限的原因，它们非常好：它们非常重要，可以对做出的决策产生巨大的影响，无论大小。让我们来讨论这些。

11.1.2 直观地与数据进行交互

我们从小就被教会从认知的角度与现实世界中的数据交互，也就是说我们更专注于分析数据而不是从整体上理解数据。我们并不善于以创造性的、有趣的方式来看待数据并去洞察

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

数据。然而现如今我们有了很大的发生变化，在数据科学和分析领域出现了一些全新的学科，特别是有些学科着重讨论与数据交互的视觉交互效果，因为视觉交互对于有些没有特定专业能力的人有能够更直接表达数据的效果。在大型项目中，项目经理也会发现视觉交互非常有趣，原因是：

- 图形化界面可以让人们一目了然地获取关键信息。对于海量数据，我们可以在短时间内筛选出什么是重要的、什么是不重要的。
- 一张图胜过千言万语。图形化界面通常更容易理解，也更容易记住。

如果想要了解更多关于数据可视化的知识，我建议你阅读 Edward R. Tufte 的图书（见图 11-1）以获取更多详细信息。

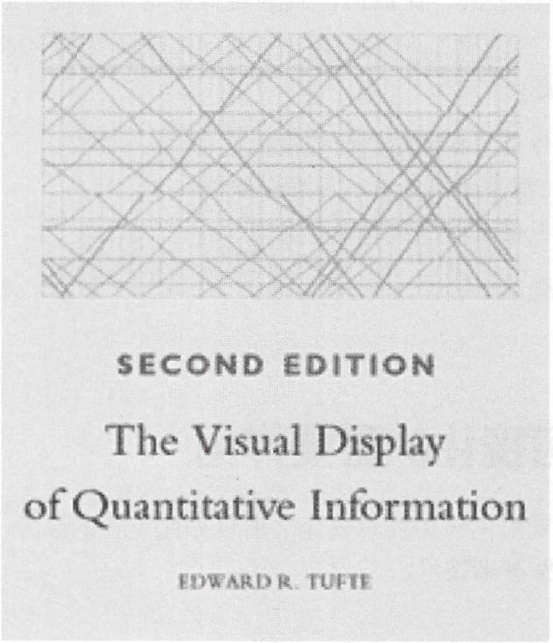


图 11-1　Edward Tufte 在数据可视化方面的相关工作

通过上面的讨论足以说明图数据可视化是很重要的，并且今天的大多数 Neo4j 项目都会提供某种形式的可视化组件。

11.1.3　模式识别

除了上节介绍的内容外，可视化还有一个重要优势就是能方便地与 Neo4j 进行交互。事实上，图数据可视化使我们可以充分使用大脑未被充分利用的能力：模式识别。当我们以传统方式（通过列表、表格、结构化报告和汇总统计）与数据交互时，通过这种模式去理解数据我们总感觉很困难，我们需要阅读和认知所有数据，然后才能完全理解数据的整体模式。但是，对于图数据可视化，在很多情况下，我们可以利用我们已经构建到计算机中的功能：模式识别。模式识别是人类大脑的关键能力，我们应该在商业、生活中利用模式识别。经常

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

以可视化的方式来表示数据，这对我们来说是自然而然的事情，并且可以在我们工作、生活中起到很大的作用。毕竟，这就是为什么在与同事一起讨论工作时，我们经常使用白板、思维导图或涂鸦，难道不是吗？

11.1.4　发现重点

图数据可视化的重要作用是可以将特殊的数据剥离出来，使我们能够在数据集中找到重要的东西。有时候这些重要的东西可能是数据实体之间的连接关系，有时候则可能是数据实体本身。如果没有图数据可视化，我们想要找到这样的数据恐怕并非易事。

出于上述原因，图数据可视化是非常重要的，任何对 Neo4j 图数据库感兴趣的人都应该仔细了解它。

11.2　图数据可视化的基本原则

为了实现图数据结构的可视化，我们需要考虑多个方面。与其他形式可视化一样，某个数据节点很容易在海量节点图中消失——正如我们经常谈论的毛球效应（见图 11-2）。例如，图的连接结构很容易变得对用户非常不清楚和不透明。我们需要一些非常明确的机制以易懂的方式向用户展示图。更具体地说，在接下来的内容中，我们将讨论三种不同的技术解决方案所采用的机制。

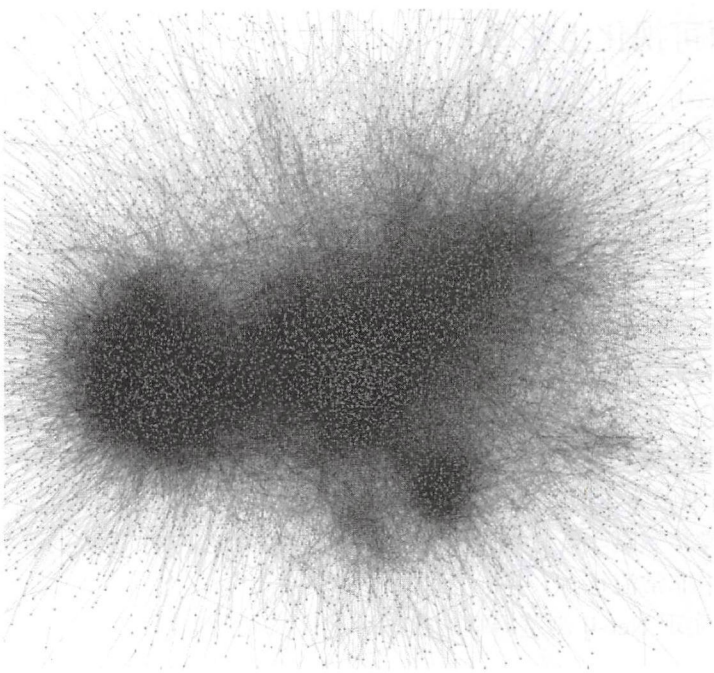


图 11-2　图数据可视化的毛球效应

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

在此之前，我要特别感谢阿利斯泰尔琼斯富有洞察力的观点，他的观点为我在本节中引出了许多想法。如果你想了解更多信息，请访问 www.apcjones.com。

下面就是三种不同的技术解决方案所采用的机制：

- 权重机制：为了使图数据可视化效果更加全面、形象，我们需要将一些相互关联密切的数据一起展示出来。很多可视化技术方案采用权重机制来将数据节点相互吸引在一起，这些权重是根据存储在 Neo4j 数据库管理系统中的节点和关系的特征计算出来的。
- 互斥机制：按照同样的逻辑，需要有一种能够保持数据节点彼此分开的机制。我们称为互斥机制，正如我们在物理课堂上学的，具有相似正电荷或负电荷的物质将相互排斥。同样，我们会将图中相互关联不大的数据节点相互分离开来，以防止它们重叠并确保它们可视化。
- 动态机制：与其他数据可视化解决方案一样，图数据可视化如果能以动态的方式呈现，就将更加生动、形象地展现数据。在 Neo4j 数据库操作界面上，数据节点不是静态地放置在某一个固定位置，而是可以在屏幕上随意移动，用户可以任意拖曳、移动。

具备以上三种机制的可视化解决方案就具有非常有趣、生动、形象的特点了。在下面的内容中我们将介绍几种不同的 Neo4j 可视化解决方案。

11.2.1 开源可视化组件库

许多使用 Neo4j 作为图数据库管理系统的开发人员都有一些非常特殊的需求：希望可以将网状的可视化界面呈现给用户并将此功能作为应用程序的一部分。这些 Neo4j 开发人员不会自己去实现这个可视化功能，他们通常喜欢使用第三方组件库来构建可视化功能。下面让我们来了解几款比较流行的可视化组件库。

1. D3.js

D3 是一种提供数据驱动文档功能的组件库。它是一个基于数据处理文档的 JavaScript 库。我们可以在 www.d3js.org 上找到最新版本。D3 使我们能够操作 HTML、SVG 和 CSS 来实现数据的可视化。D3 能够在目前多款主流的浏览器上运行，并且不依赖其他组件库，D3 将强大的可视化组件与数据驱动相结合，以处理 HTML 和 XML 为基础的（DOM）文档。

D3.js 不仅用来做图数据的可视化，还旨在解决许多不同类型数据的可视化问题，并且可以与用户交互来操作数据集。它在用户与数据的交互方面提供了很大的灵活性。因此 D3 组件库在开发人员中非常流行。

D3.js 被用来实现 Neo4j 浏览器端的数据可视化（见图 11-3）。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

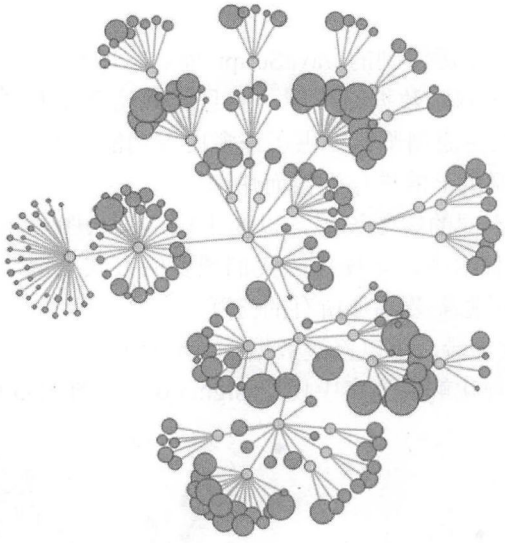


图 11-3　D3.js 的图数据可视化

2. GraphViz

GraphViz 是一个开源图可视化组件库，我们可以从 www.graphviz.org 下载。GraphViz 被认为是可视化组件的鼻祖，但这个组件库依然持续地更新和完善。它提供了几个主要的布局功能，包括交互式图形界面、常用帮助工具、库和语言绑定功能。GraphViz 的开发团队声称无法将大量精力投入到 GUI 编辑器的开发中，但有很多第三方项目实现了 Graphviz 的 GUI 编辑器。

Graphviz 以一种简单的语言去描述图结构并将图结构以某种形式展示出来，比如，在网页上用图片、SVG、PDF 或者 Postscript 等。

Graphviz 具有许多用于展示图（见图 11-4）所需的特有功能，例如颜色、字体、表格节点布局、线条样式、超链接和自定义形状等。

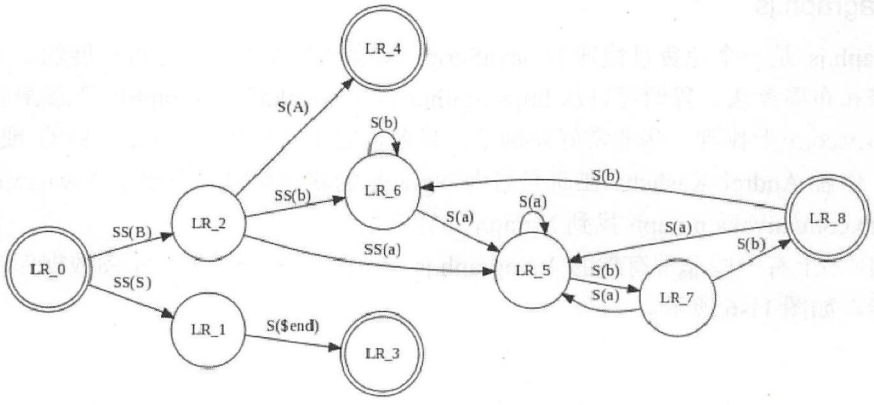


图 11-4　GraphViz 图数据可视化

接下来让我们看一下 Sigma.js。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

3. Sigma.js

Sigma.js 是一个专门用于图绘制的 JavaScript 库。它的官方网站是 www.sigmajs.org。它可以方便地在网页上绘制出网状数据结构图，并且能够让开发人员很方便地将它集成到前端应用项目中。Sigma 为现在主流浏览器提供了许多内置功能，如 Canvas、WebGL 渲染器、鼠标和触摸支持。这使得在网页上的操作流畅而快速。

Sigma.js 提供了许多不同的参数设置，以便于自定义绘图功能和交互功能。我们也可以直接将自己的函数添加到脚本中，以便按照我们想要的方式呈现节点和关系边。Sigma.js 是一个渲染引擎，我们可以根据需要添加所有的交互功能。

Sigma.js 的 API 非常灵活，可以修改数据、移动视角、刷新渲染、事件监听等。我们将在本章后的 Línkurio.us 可视化解决方案中使用 Sigma.js（见图 11-5）。

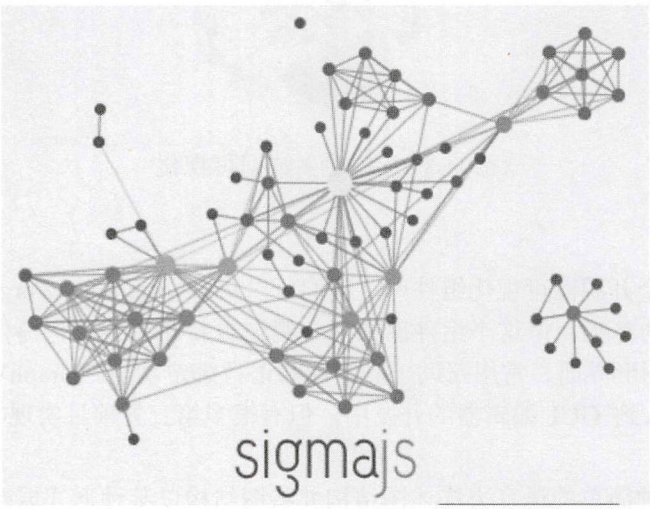


图 11-5　Sigma.js 图数据可视化

接下来让我们看一下 Vivagraph.js。

4. Vivagraph.js

Vivagraph.js 是一个免费且快速的 JavaScript 图绘制组件库。它是可扩展的，并支持不同的渲染引擎和布局算法。我们可以从 https://github.com/anvaka/VivaGraphJS 下载最新版本，并在 www.yasiv.com 上找到一些非常好的例子。目前，它支持使用 WebGL、SVG 或 CSS 格式的渲染图。作者 Andrei Kashcha 在研究名为 Ngraph 的软件库时开发出了 Vivagraph，可以在 https://github.com/anvaka/ngraph 找到 Ngraph 的详细信息。

Yasiv 网站上有一些非常有趣的 Vivagraph.js 示例。有些例子展示了图数据库在商品推荐方面的功能，如图 11-6 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

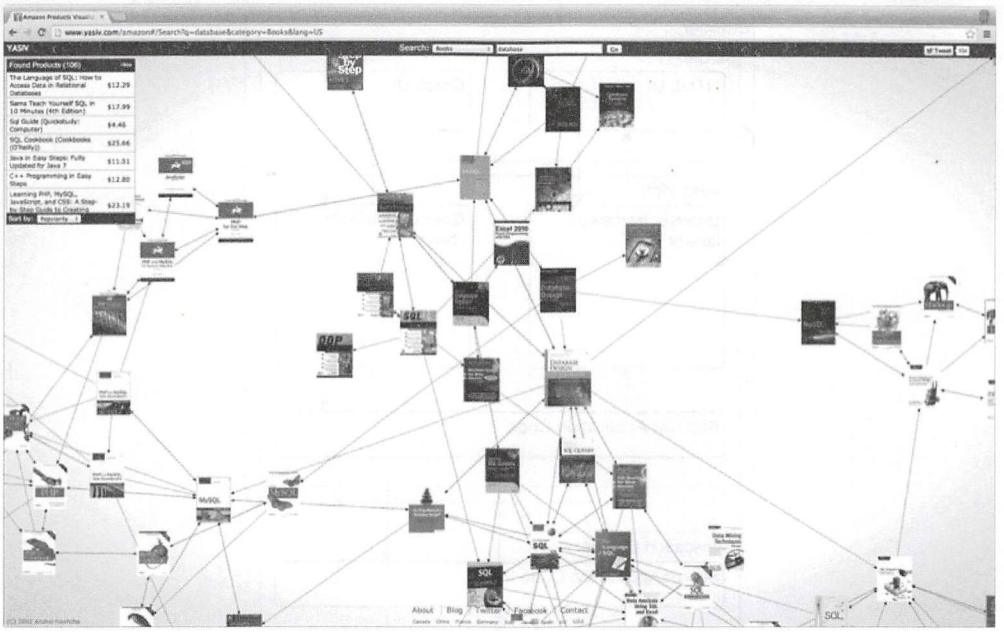


图 11-6　Vivagraph.js 的可视化

下面让我看看 yWorks。

5. yWorks

yWorks yFiles for HTML 是一个功能强大的组件库。我对它的自动布局功能印象特别深刻。

它的主要布局功能如下：

- 层级布局
- 有机化布局
- 正交化布局
- 树状布局
- 圆形布局
- 气球形布局
- 径向布局
- 平行布局
- 路由布局

yWorks 用法相当简单：使用提供的页面模板，然后在脚本中写入图控制代码。

上面介绍的就是目前比较流行的可视化组件库。

11.2.2　将可视化组件库应用到项目中

将可视化组件库集成到项目中常遵循如图 11-7 所示的模式。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

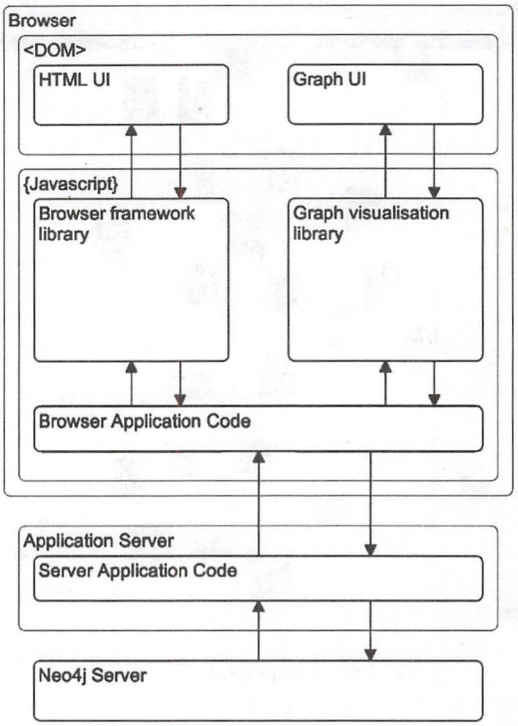


图 11-7　将可视化组件库集成到项目中

浏览器将包含两部分：

- DOM 和实际数据。图数据将成为构建整个网页的一部分。
- JavaScript，包括前端框架以及图可视化组件库以及其他一些应用程序代码。

我们项目的服务器是在网络的另一端，服务器响应用户请求并将图界面呈现给用户。服务器通常会运行着一些业务逻辑，并且可能必须与 Neo4j 等数据库管理系统集成在一起。

在接下来介绍的 Neo4j 服务器上程序将根据上述业务逻辑生成查询结果并返回结果集。采用上述架构的好处是，我们可以很方便地将可视化组件库集成到我们的项目中。

下面让我们转向一些可与 Neo4j 结合使用的可视化解决方案。

11.2.3　可视化解决方案

我们在此将可视化组件库和可视化解决方案区分开来，原因是它们的用途非常不同。

Neo4j 许多用户一直在开发特定于自己项目的可视化应用程序，这些应用程序通常是特殊定制的。他们的目标是解决一个非常具体的可视化问题，并使用某些组件库来做到这一点，利用这些组件库虽然可以根据自己的喜好来定制自己的可视化方案，但这样也会付出一些工作量。

当然，也有 Neo4j 用户只需要一般的、优化程度较低但更容易使用的可视化解决方案。对于这些用户来说，整套的可视化解决方案是一个更好的选择，因为它们是高可用、全封装、现成的解决方案，只需要很少或根本不需要额外编码就可以拿来使用。接下来让我们讨

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

论这样的解决方案。

注意，我们将按字母顺序来讨论这些解决方案。我们并不讨论哪个解决方案更好，因为这些工具中的大多数功能是互补的。

1. Gephi

Gephi 是一个常用于复杂系统中显示动态图结构的、交互式的、可视化平台。它由 Gephi Consortium 开发，属于一个法国非营利机构，这个机构负责维护 Gephi。他们的目的是开发多款工具来构建开源生态系统，此机构专注于对大型网状数据进行实时可视化展示和实时分析。Gephi 类似 Photoshop 但它是专门处理数据的：Gephi 帮助数据分析师直观地了解数据的整体模式和变化趋势，便于找到数据的特殊变化。

该工具可以在 Windows、Linux 和 Mac OS X 系统上运行，并具有非常可扩展的特性（NetBeans 平台插件），并且允许我们导入和导出 Neo4j 数据库。

在我看来，Gephi 有趣的是它能够同时进行图可视化和图数据分析。利用它我们可以很容易地看出数据的整体模式，如图 11-8 所示。

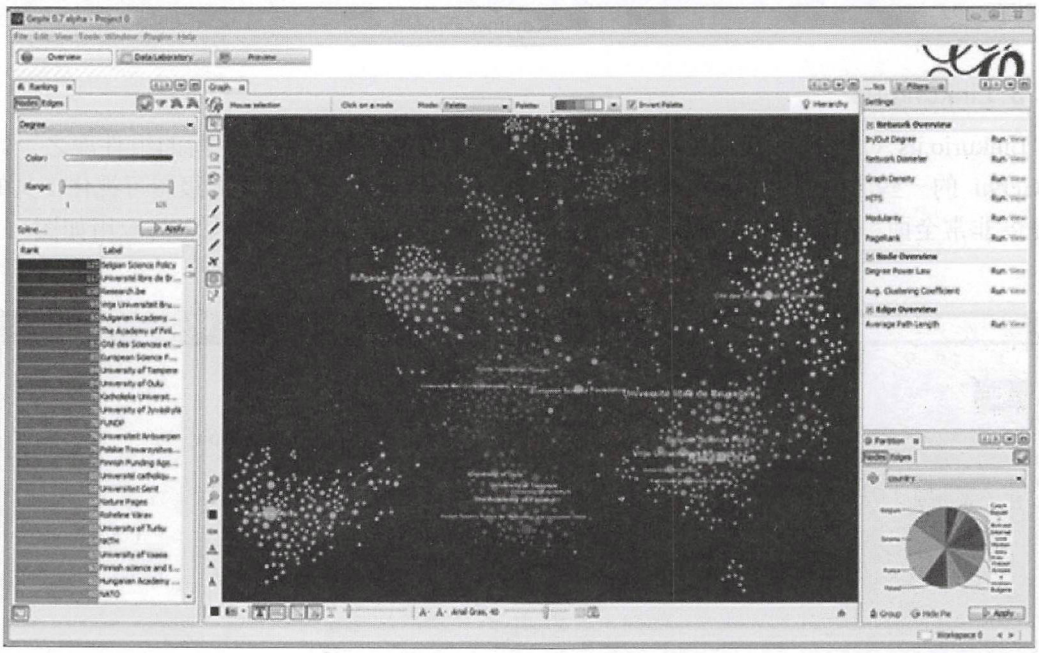


图 11-8　Gephi 的应用

2. Keylines

位于英国的初创公司 Cambridge Intelligence（http://cambridge-intelligence.com/）一直在构建一个非常有趣的图可视化工具，称为 Keylines（http://keylines.com/），它已经应用在执法、欺诈检测、反恐、CRM、销售和社交领域。

Keylines 的图可视化（见图 11-9）：从技术角度来看，Keylines 在许多方面都很有趣，主要是因为它可以部署在任何浏览器或设备上，而且架构和支持方面没有太多差异。它允许标准的“开箱即用”部署，也可以深度定制。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

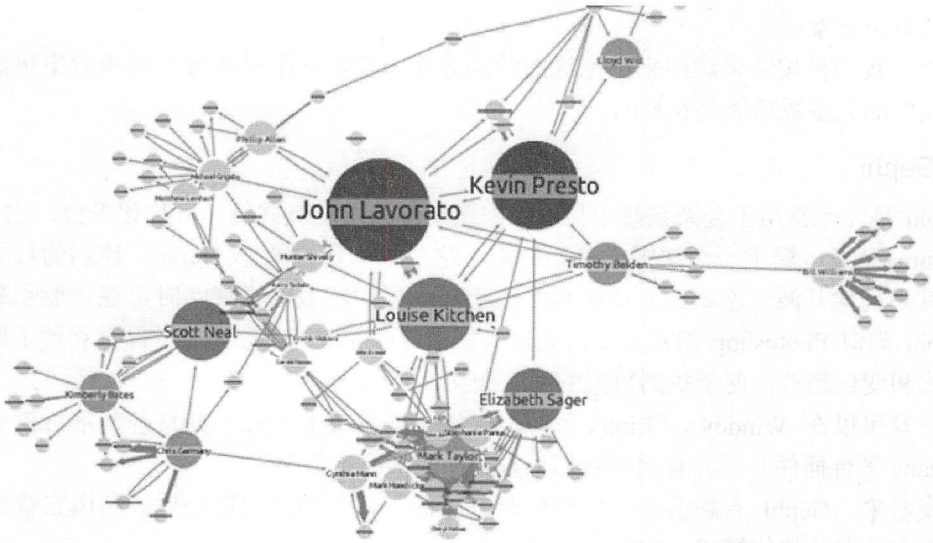


图 11-9　Keylines 的应用

3. Linkurio.us

Linkurio.us（http://linkurio.us/）可能是图可视化领域中更有趣和更新的组件库之一。它由 Gephi 的一些原创始人和贡献者在法国创建。他们现在可以为需要高级图可视化工具的人们提供非常全面、价格适中、易于访问的程序接口。Linkurio.us 图可视化的应用如图 11-10 所示。

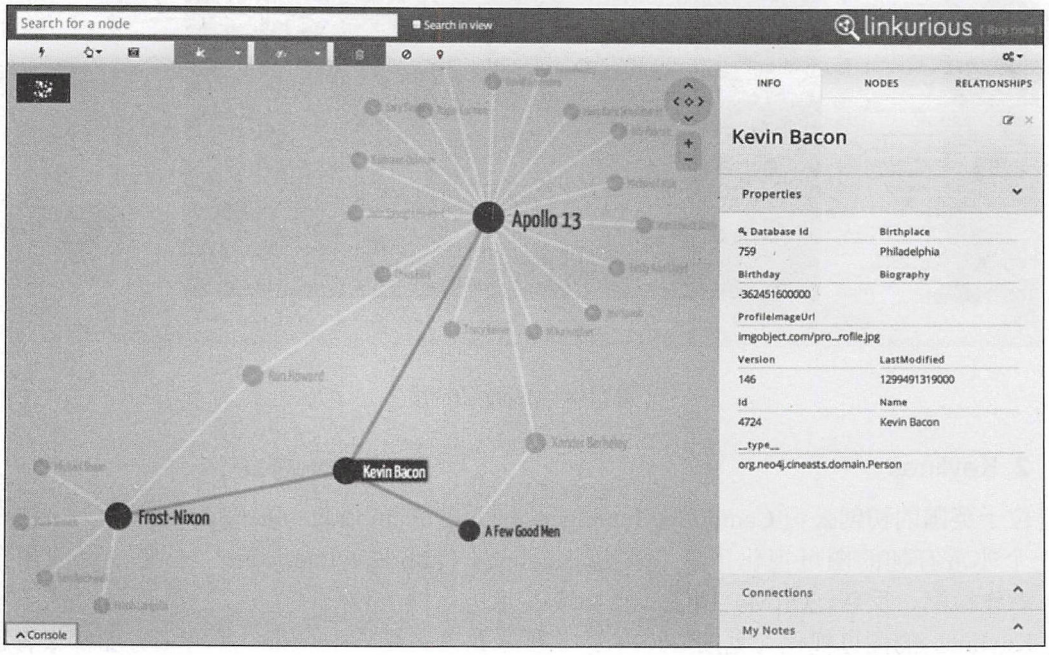


图 11-10　Linkurio.us 图可视化

非卖品，仅供非商业用途或交流学习使用
严禁网络传播本PDF，违者责任自负！

Linkurio.us 是图可视化领域比较不错的一个解决方案，其中包含许多有趣的功能，并且可以在现有的 Neo4j 数据库管理系统之上进行即插即用的安装。

4. Neo4j 浏览器

随着 Neo4j 2.0 在 2013 年末的到来，Neo Technology 团队提供了一种与图数据交互的新方式，也就是使用 Cypher 与图数据库进行交互的直观方式：Neo4j Browser。Neo4j Browser 既是数据查询工具也是一种开发环境。

Neo4j Browser 提供了许多在图数据交互中常用的工具（见图 11-11），可以保存查询结果和图的展示样式，并且它是可扩展的。

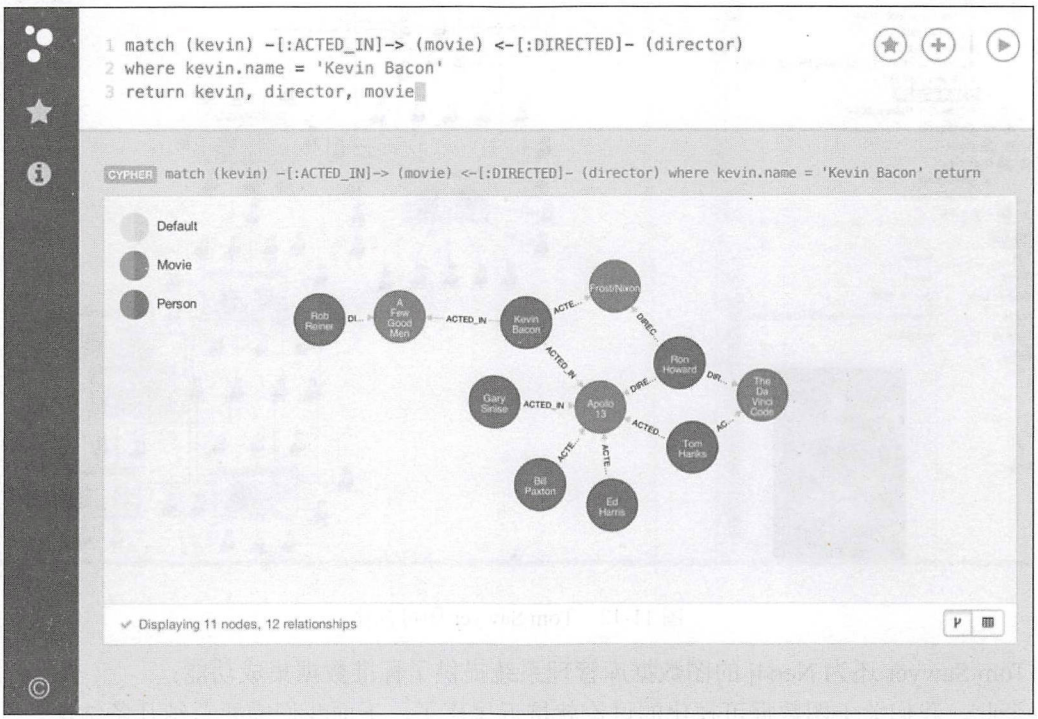


图 11-11　Neo4j Browser 可视化

在本书撰写期间 Neo4j Browser 仍在开发中，并且在不断添加新功能。

更有趣的是 Neo Technology 允许将浏览器可视化功能嵌入到我们自己的应用程序中。

5. Tom Sawyer 图可视化

在欧洲，图数据可视化领域有一个鲜为人知的方案——Tom Sawyer Software 的 Perspectives 系列产品。它的官方网站是 https://www.tomsawyer.com。关于 Perspectives 更详细的信可息参见 https://www.tomsawyer.com/products/perspectives/index.php。

这是一个基于图的高级软件项目，用于构建企业级的数据关系可视化和数据分析应用程序。它具有完整的**软件开发工具包**（SDK）和基于图的设计和预览环境，使我们可以更轻松地构建、测试和部署图可视化项目。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

Tom Sawyer 图可视化案例如图 11-12 所示。

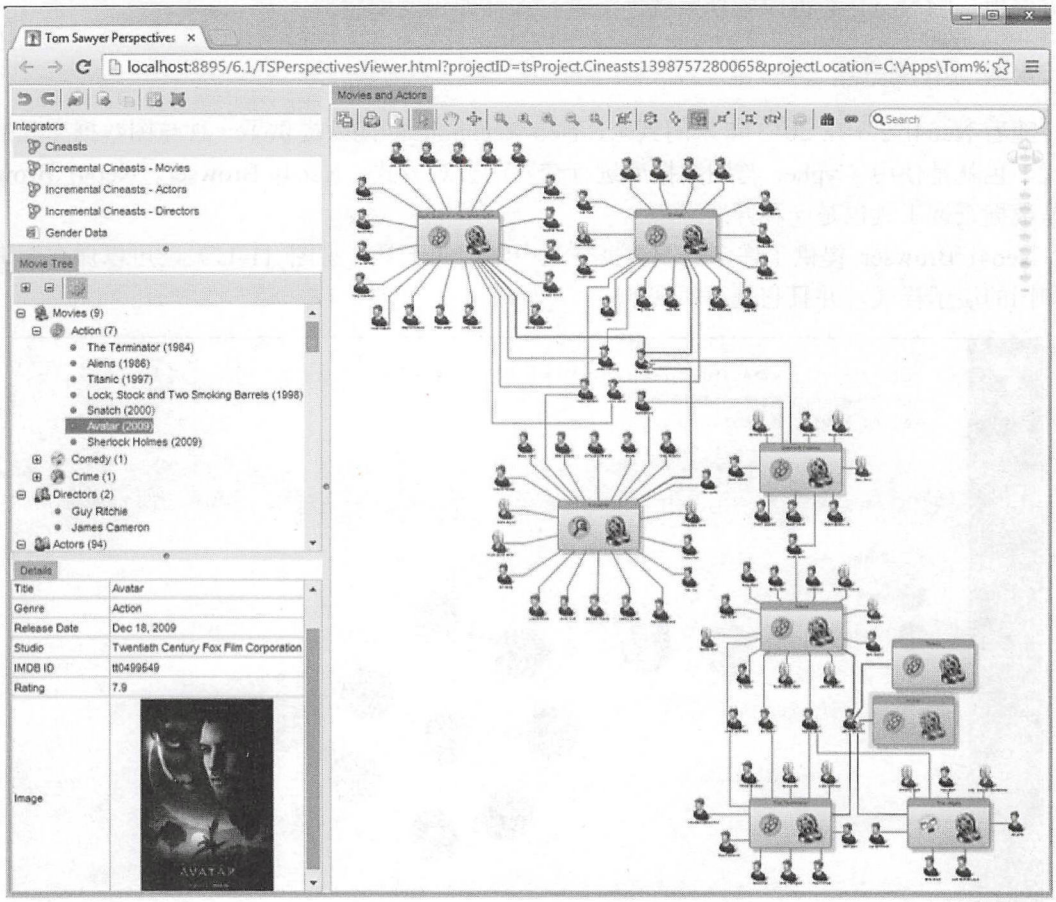


图 11-12　Tom Sawyer 图可视化

Tom Sawyer 还为 Neo4j 的图数据库管理系统提供了标准数据集成功能。

至此，我们关于图数据可视化的讨论就接近尾声了，下面我们需要总结几条结论。

11.2.4　问题和陷阱

现在应该清楚，一旦我们使用 Neo4j 这样的图数据库管理系统来存储、处理数据，我们的项目就注定要与图可视化紧密结合。我们可以从图可视化学到很多东西，但是要指出两个注意事项，在参与可视化项目时应始终牢记在心。

1. 烟花效果

虽然图可视化使得用户体验非常好，但是我们作为提供接口与数据进行交互功能的 IT 开发人员也必须意识到这些可视化界面有时候显得非常杂乱。

我们称之为烟花效果，虽然其他的数据可视化也会出现这种现象，但是我们要注意到这一点很重要，并不可能所有人都喜欢可视化的效果。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

可视化的目的应始终是与观众进行交流，并将信息从我们在图数据库管理系统（例如 Neo4j）上构建的软件应用程序呈现给用户。

这似乎是一个微不足道的问题，但我们从经验中知道它经常被人遗忘。要演示这种效果非常简单，只需点击多个关系中的多个节点即可。图视图中的节点、关系比我们在第 8 章将数据导入 Neo4j 中加载的内容更多。几次双击之后，屏幕上充满了节点，这样整个结果集界面几乎不可读。

2. 加载效果

另外一个在图数据可视化方面比较重要的问题就是“加载效果”。数据展示界面的动态加载效果并不是图数据可视化专有的问题，其他类型的数据展示也会有这样的问题。加载效果就是数据界面在还没有完全加载到浏览器端时显示的效果。

我们一般会按照用户为某些数据元素的配置来呈现数据视图，比如颜色、布局。这样数据的呈现效果就会符合用户的期望。

11.2.5　Cytoscape 实例

在接下来的内容中我们来看下一个使用名为 Cytoscape 组件库做的实例。

Cytoscape 是基于 MIT 协议的开源项目，源码可以在 GitHub（https://github.com/cytoscape/cytoscape.js）下载，文档可以在 http://js.cytoscape.org/下载。

这个用例的目的是从数据库中提取数据并展示出来。这些数据是展示 JavaScript 驱动用例在 UNIX 类操作系统上的支持情况。

这个用例运行的效果图如图 11-13 所示。

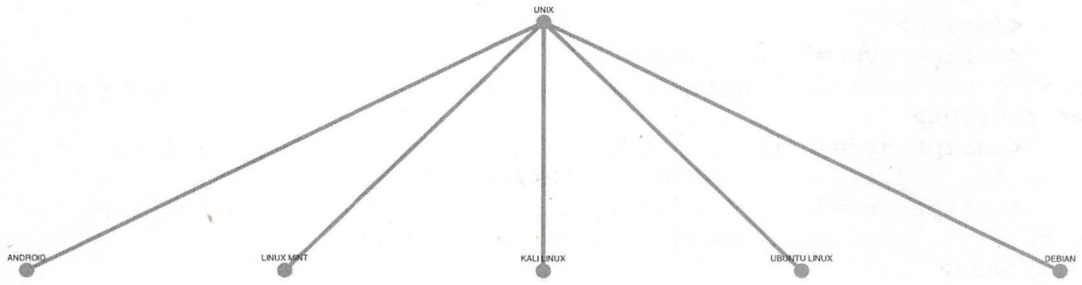


图 11-13　Cytoscape 运行效果

虽然编写的 JavaScript 代码很简单，但我依然尝试着将代码全部编写进一个文件中。下面让我们看看实例代码。

首先，我们使用 Cytograph 的样式设置功能设置要显示的图的大小尺寸。从 CND 加载 JavaScript 库、创建一个 div 元素，再用 Cypher 查询数据库并创建代表 UNIX 类操作系统的节点。这些节点是在一个循环语句中创建的。因为 Cytoscape 中没有相同节点合并的功能，所以我们不能为所有节点都使用同一个预定义对象以防止创建相同的节点。为了保险，我们

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

会在循环语句中检查生成的节点。

这样，图节点使用数据库中的数据作为数据源。

然后，我们使用 JavaScript 驱动库连接到本地的 Neo4j，连接字符串为“neo4j/password”，其中 password 使用密码明文，当然，在实际生产环境中我们是不提倡这样做的。我们的代码将请求 Neo4j 的 REST 接口。

当我们的 JavaScript 代码在页面中载入后就会发出查询请求，然后按照返回的结果集创建 JavaScript 对象，返回的结果集（JSON）的结构应该是符合 Cytoscape 要求的，代码中的 then 是一个 JS 关键字，用来执行一系列的函数。then 也通常被用来执行一些定义复杂的、用来绘制布局的公用方法。

接下来，Cytoscape 就开始不断创建节点，然后整个布局就展示出来了。

我们推荐大家阅读一下文档并且亲自运行代码。

这个实例的源代码如下：

```
< !DOCTYPE html>
  <html lang="en">
    <head>
      <meta charset="utf-8">
      <meta http - equiv="X-UA-Compatible" content="IE=edge">
      <meta name="viewport" content="width=device-width, initial-
scale=1">
      <title>Neo4j + Cytoscape  < /title>
      <style>
      #cytograph {
          width: 90%;
          height: 90%;
          position: absolute;
          top: 0px;
          left: 0px;
      }
      </style>
      <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js
">< /script>
      <script src="https:/ / cdnjs.cloudflare.com / ajax / libs /
cytoscape / 3.1.4 / cytoscape.js "></script>
      <script src="https: //cdn.rawgit.com/neo4j/neo4j-javascript-
driver/1.1/lib/browser/neo4j-web.js ">"</script>
    </head>
    <body>
      <h1>Basic cytoscape.js use with neo4j</h1>
      <div id="cytograph "></div>
      <script>
      var fam; // first object is unix family.
        fam = new Object();
        fam.data = {}
        fam.data['group'] = 'nodes';
        fam.data['id'] = 'UNIX';
        fam.data['name'] = 'UNIX';
        var neo = neo4j.v1;
```


非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

```
        var driver = neo.driver("bolt: //localhost",
neo.auth.basic("neo4j", "password"));
        var session = driver.session();
        var neoElements = [];
        neoElements.push(fam);
        $(document).ready(function() {
          session.run('MATCH (n:Os)--(f:OsFamily) RETURN n.name AS
nname,f.name AS fname').then(function(result) {
            var i = 1;
            result.records.forEach(function(r) {
              var nouveau = new Object();
              nouveau.data = {}
              nouveau.data['group'] = 'nodes';
              nouveau.data['id'] = '' + r._fields[0];
              nouveau.data['name'] = '' + r._fields[0];
              var rel = new Object();
              rel.data = {}
              rel.data['group'] = 'edges'rel.data['id'] = r._fields[0] +
r._fields[1];
              rel.data['source'] = '' + r._fields[0];
              rel.data['target'] = '' + r._fields[1];
              neoElements.push(nouveau);
              neoElements.push(rel);
            });
          }).then(function populate(result) {
            var options = {
              name: 'breadthfirst',
              fit: true,// whether to fit the viewport to the graph
              directed: false,// whether the tree is directed downwards
(or edges can point in any directionif false)
              padding: 30, // padding on fit
              circle: false,// put depths in concentric circles if true,
put depths top down  if false
              spacingFactor: 1.75, // positive spacing factor, larger =>
more space between nodes(N.B.n / a if causes overlap)
              boundingBox: undefined,// constrain layout bounds; { x1,
y1,x2,y2}or {x1,y1,w,h}
              avoidOverlap: true,// prevents node overlap, may overflow
boundingBox if not enough space
              nodeDimensionsIncludeLabels: false, // Excludes the label
when calculating node bounding boxes for the layout algorithm
              roots: undefined,// the roots of the trees
              maximalAdjustments: 0, // how many times to try to position
the nodes in a maximal way(i.e.no backtracking)
              animate: false,// whether to transition the node positions
              animationDuration: 500,// duration of animation in ms if
              enabled animationEasing: undefined,// easing of animation
if enabled
              ready: undefined,// callback on layoutready
              stop: undefined // callback on layoutstop
            };
            var cy = cytoscape({
              container: document.getElementById('cytograph'),
```

严禁网络传播本PDF，违者责任自负！

```
          elements: [],
          style: cytoscape.stylesheet().selector('node').style({
            'content': 'data(id)'
          }).selector('edge').style({
            'target-arrow-shape': 'triangle',
            'width': 8,
            'line-color': '#00ff00',
            'target-arrow-color': '#ff0000'
          }).selector('.highlighted').style({
            'background-color': '#61bffc',
            'line-color': '#61bffc',
            'target-arrow-color': '#61bffc',
            'transition-property': 'backgroundcolor,
line-color, target-arrow-color',
            'transition-duration': '0.5s'
          })
        }); cy.add(neoElements);
        var leLayout = cy.layout(options); leLayout.run(); cy.fit();
      });
    });
< /script>
</body >
</html>
```

11.3 问答

问题 1：如果我想构建一个图的可视化程序，就必须从头开始创建可视化组件库吗？

A. 是的

B. 不是

正确回答：不是。虽然许多应用程序从自己定制的可视化解决方案中受益，但有许多解决方案和库可以帮助我们。至少，这些库和或解决方案将为我们提供一些便利。

问题 2：3 个基本的图可视化组件库是什么?

```
Gravity, Obesity, Charge
Springs, Gravity, Charge
Charge, Gravity, Verbosity
```

正确答案：Springs, Gravity, Charge。

问题 3：大多数图可视化使得用户无法看懂、显得杂乱无章的现象是什么？

A. 森林效果

B. 载入效果

C. 烟火效果

答案是：烟火效果。

严禁网络传播本PDF，违者责任自负！

11.4 小结

这一章我们讨论了图可视化的相关内容。在本章中，我们探讨了图形可视化对现代数据应用程序影响重大的原因。我们还介绍了可供应用程序开发人员用来创建其图数据库应用程序的不同工具和技术。然后，我们指出可视化解决方案中一些非常重要的副作用以及我们在参与这样的项目时如何考虑这些影响。

下一章将我们将介绍数据重构，因为我们知道，第一次创建的数据模型并不总是完美的，并且模型可能会随着时间的推移而变化。

严禁网络传播本PDF，违者责任自负！

第 12 章

◂ Neo4j数据重构 ▸

本章将专注于讨论图数据模型的演变。

我们的数据模型随着时间的推移会演变。这可能是因为在一开始我们的数据模型不是完美的，而最可能是因为不断变化的需求。我们接下来将从简单的例子开始，然后逐步推出复杂的例子。

本章我们将讨论如下主题：

- 捕获关键定位/位置的信息
- 对定位/位置进行重建模
- 从定位/位置抽取信息
- 添加和获取位置信息

12.1 预备步骤

大家肯定知道这个谚语：安全胜过遗憾和墨菲定律，不是吗？

有个建议很简单：备份好我们的数据（详见附录章节有关备份服务器的方法）。

在做结构性改变之前，我们应先保证库的安全。这对任何数据库都有效。而且，在完成之前，备份过程是一个存亡未知的状态。

12.2 简单改变

我所说的简单改变是指对单个节点的改变，不包含与之关联的节点和关系。

12.2.1 重命名

Cypher 里没有重命名的功能，但 APOC 里有这样的过程。

这句“APOC 里有这样的过程”在 Neo4j 社区版里几乎成了圣歌。

早期版本结尾时说过——如果没有那些，请找 Michael，Michael Hunger，我已经写过有

严禁网络传播本PDF，违者责任自负！

关他的情况。

通过 APOC，才有可能去重命名标签、关系、节点属性、关系属性。

目前版本中，关系名为 type，所以我们这样调用过程：

```
CALL apoc.refactor.rename.label(oldLabel, newLabel, [nodes])
CALL apoc.refactor.rename.type(oldType, newType, [rels])
CALL apoc.refactor.rename.nodeProperty(oldName, newName, [nodes])
or
CALL apoc.refactor.rename.typeProperty(oldName, newName, [rels])
```

我们可以选择提供节点或关系的列表。

12.2.2　添加数据

添加数据将使当前数据模型充实一些新的属性。这相当安全。没有任何破坏性改变的风险。

1. 使用默认值添加数据

我们再次使用有关高塔（Tower）的例子，我们想知道它们上面是否开了一个餐馆。一个 yes/no 字段完美适用 Boolean 属性。我们将调用这样的查询，假定建造一个不适合开一家旋转餐厅的高塔是一种浪费：

```
MATCH (n:Tower)
SET n.hasRestaurant = toBoolean("true")
RETURN n
```

请注意里面生成 Boolean 值的转换函数，这样它允许简化下列查询：

```
MATCH (n:Tower)
WHERE n.hasRestaurant
RETURN n
```

2. 使用指定值添加数据

我们可能需要对每一个指定节点指定一个值，有两种主要情况：要么值是已知的，要么值是需要计算出来的。如果值是已知的，我们肯定能从数据源读取到，比如一个 JSON 文件、一个 XML 文件或者一个 CSV 文件。

在前两种情况下，APOC 可以帮助我们，比如可以用 LOAD CSV，就像在“第 8 章 Neo4j 数据导入”那样。方法如下：

```
LOAD CSV OR CALL apoc.load.json(url) YIELD value
MATCH the node with its ID
SET the new property to the given value
```

如果值未知且不得不从已知数据里计算出来，就需要一些技巧了，因为可能需要转换函数从一种字段类型转换成另一种。举一个例子：从一个地址里获取街道号码，且通过 toInt 方

法转换成数值字段存储下来。方法如下：

```
MATCH the node to change and all useful nodes and relations
SET newProperty = computed value
```

3. 检查我们的值

在任何情况下，我们都不能确定这种情况：hasRestaurant 属性存在于现有的所有 Tower 节点中。幸运的是，Cypher 考虑到了这点，因为它才有可能发现未完成的节点，那就是 NOT exists(node.property) 语句，就像这样：

```
MATCH (n:Tower)
WHERE NOT exists(n.hasRestaurant)
RETURN n
```

然后，我们就可以标记这些节点为脏数据（使用我们选择的标签），通过这个标签，我们就可以决定是否从查询中排除它们。

通过这些节点，我们可以找到属性值未被设置的原因，并可以进行改进处理，这个我建议大家编写一个函数（或过程）。

从某方面讲，我提醒大家或许可以定义一个约束条件，以便一种标签隐含一个属性的存在：

CREATE CONSTRAINT ON (t:Tower) **ASSERT** exists(t.hasRestaurant)

12.2.3 删除数据

从节点删除属性就像下面这样简单：

```
MATCH n
DELETE n.property
```

要确保我们的 MATCH 命令使用标签过滤了，否则只有面对数据丢失的悲惨结果了。（但是大家会有一个备份，不是吗？）和添加属性到我们的节点不同，这会对我们应用程序的查询产生影响。我们将在后面的段落中介绍。

12.3 重大改变

当要进行重大改变时，我们要准备充分。套用哲学上的说法就是：我在哪里，我去哪里。

12.3.1 知道我们的模型

知道我在哪里就是知道我的实际模型。早期 APOC 插件里有一个功能，就是用来以图的

方式展示我们的元数据模型，现在成为 Neo4j 和 Cypher 的一个默认功能：

```
CALL db.schema()
```

因为它是我们的模式的即时视图，所以建议大家保存这个模式的展示结果（见图 12-1）。

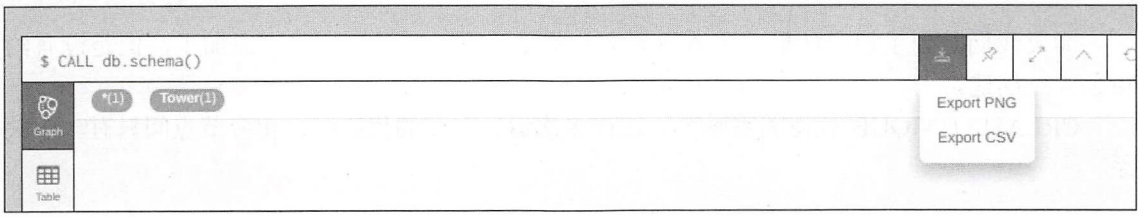


图 12-1　如何导出模式的视图

导出两种格式，我们可以用 PNG 图片作为新模型的基础，同时 CSV 列出所有属性。

接下来我们将把要做的改变以全景图的形式展示给大家。

12.3.2　重构工具

为了重构我们的图数据库，我们有两种工具可用。第一种是标准的 Cypher，第二种是 APOC——尤其是过程库里的 apoc.refactor 包。再提一下，强制要求是专心和数据备份，不过大家应该已经知道了。

12.3.3　属性变标签

基于一个属性给节点添加一个标签，有两个步骤：

- 添加一个标签给节点。
- 删除这个属性。

举例来说，如果我们想加一个 hasRestaurant 标签给带有餐饮的高塔（Tower），查询如下：

```
MATCH (t:Tower)
WHERE t.hasRestaurant
SET t:HasRestaurant
REMOVE t.hasRestaurant
RETURN t
```

这也说明了命名习惯的必要性：HasRestaurant 是一个标签，因为它以大写字母开头；而 hasRestaurant 是一个属性，因为它以小写字母开头。

12.3.4　属性变节点

现在，要从节点里分离出一个属性以形成新的节点。例如，为了在节点间分享一个值，我们将看到如何从高塔中分离出国家数据，从其 country 属性生成新的带有:Country 标签的节

点，且使用:IN_COUNTRY 关系连接到高塔（Tower）节点：

```
MATCH (t:Tower)
MERGE (c:Country {code: t.country})
CREATE UNIQUE (t)-[:IN_COUNTRY]->(c)
REMOVE t.country
RETURN t,c
```

如果我们看到了相应结果，应该得到加分奖励，虽然一种新的节点添加了，但是没有约束条件被创建。

CREATE UNIQUE 使得关系唯一。这样多次运行上面的代码时，相应节点间只有一个关系。

12.3.5 相关节点变标签

假设我们希望根据国家（Country）节点向高塔（Tower）添加一个标签，如果标签具有静态值，则示例查询如下：

```
MATCH (t:Tower)--(c:Country)
SET t:HasCountry
```

这个例子没有什么实际意义，但它告诉我们如何指定两个节点之间的任何关系。因此，还是值得一提的。

如果标签的值是动态的，那么 APOC 可以提供帮助——我们已经看到了可以添加一个标签数组的这个过程。

下面查询将向高塔（Tower）节点添加国家代码标签：

```
MATCH (t:Tower)--(c:Country)
CALL apoc.create.addLabels(id(t), [c.code]) YIELD node
RETURN node
```

12.3.6 合并节点

将几个节点合并在一起是可以实现的。让我们先创建一些节点：

```
CREATE (a:FIRST:MERGE {propF1: 'AF1',propF2: 'AF2'}), (b:SECOND:MERGE
{propS1: 'BS1',propS2: 'BS2'}), (c:THIRD:MERGE {propT1: 'CT1',propT2:
'CT2',propT3: 'CT3'})
```

现在来看看它们：

```
MATCH (n:MERGE)
RETURN n
```

现在，让我们将其合并到第一个节点上：

```
MATCH (a:FIRST:MERGE), (b:SECOND:MERGE), (c:THIRD:MERGE)
CALL apoc.refactor.mergeNodes([a,b,c]) YIELD node
RETURN node
```

这将产生一种带有四个标签的节点以及以下属性（这是文本视图中的复制/粘贴出来的）：

```
{"propF2":"AF2","propF1":"AF1","propS1":"BS1","propT2":"CT2","propT1"
:"CT1","propS2":"BS2","propT3":"CT3"}
```

这是预期的结果。如果这些节点有同名的关系和同名的属性呢？我们来看看。我们需要删除这些节点，并使用新的规范重新创建有限的数据集：

```
MATCH (n:MERGE)
DELETE n
CREATE (a:FIRST:MERGE {propF1: 'AF1',propF2: 'AF2'}), (b:SECOND:MERGE
{propS1: 'BS1',propS2: 'BS2',propF2:'VALUE FROM SECOND'}),
(c:THIRD:MERGE
{propT1: 'CT1',propT2: 'CT2',propT3: 'CT3', propF1:'VALUE OF THIRD'})
```

如果我们像之前那样合并这些节点，结果如下：

```
{
"propF2": "VALUE OF THIRD",
"propF1": "VALUE OF THIRD",
"propS1": "BS1",
"propT2": "CT2",
"propT1": "CT1",
"propS2": "BS2",
"propT3": "CT3"
}
```

这是我们预期的结果。对于相同的属性，节点 a 的值被节点 b 的值代替，而节点 b 的值被节点 c 的值代替。

12.3.7　关系

为了说明这一段，让我们用关系重新创建数据集。

首先，让我们清除图：

```
MATCH (n:MERGE)
DETACH DELETE n
```

现在，让我们用下面的查询创建一些数据：

```
CREATE (a:FIRST:MERGE {name:'a',propF1: 'AF1',propF2: 'AF2'}),
(b:SECOND:MERGE {name:'b',propS1: 'BS1',propS2: 'BS2',propF2:'VALUE
FROM SECOND'}), (c:THIRD:MERGE {name:'c',propT1: 'CT1',propT2:
'CT2',propT3:'CT3', propF1:'VALUE OF THIRD'})
CREATE (a)-[:A2B]->(b),(b)-[:B2B]->(b),(a)-[:A2C]->(c),(b)-[:B2C]-
>(c)
```

并检查我们的小数据集：

```
MATCH (n:MERGE)
RETURN n
```

我们开始的图如图 12-1 所示。

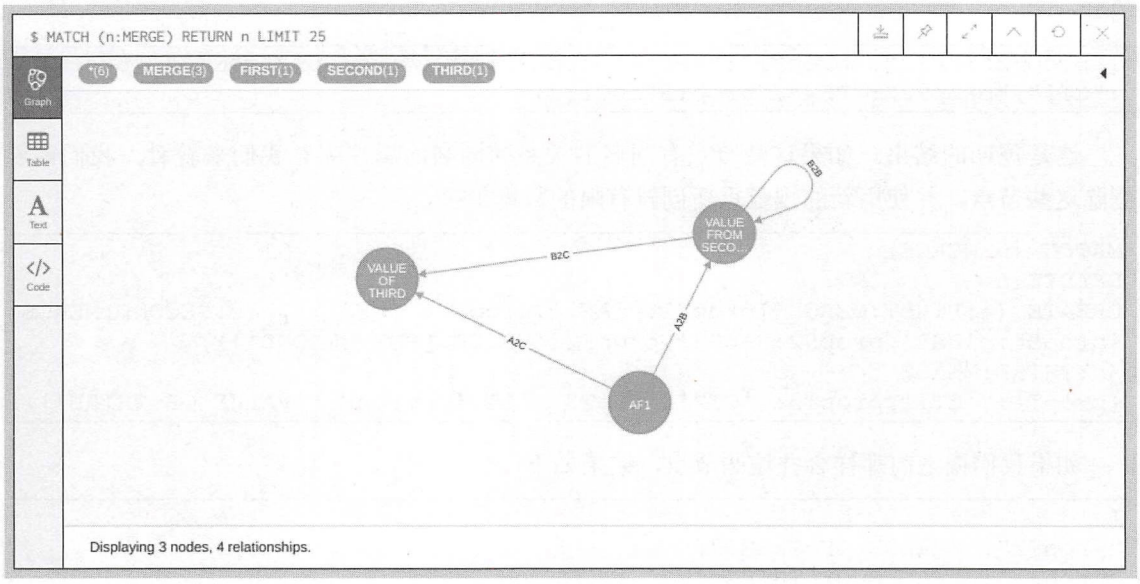


图 12-1　一个命名和导向良好的图

我们现在可以反转关系 A2B 如下：

```
MATCH (a:FIRST:MERGE)-[rel:A2B]-(b:SECOND:MERGE)
CALL apoc.refactor.invert(rel) YIELD output
RETURN output
```

用 MATCH 查询：

```
MATCH (n:MERGE)
RETURN n
```

我们可以看到，关系 A2B 现在的方向是从 B 到 A。这看上去有些混乱，因此迫切需要重新命名这个关系。要做如下操作：

```
MATCH (a:FIRST:MERGE)-[rel:A2B]-(b:SECOND:MERGE)
CALL apoc.refactor.setType(rel,'B2A') YIELD output
RETURN output
```

A2B 关系更名为 B2A。成功！

我们将继续打乱逻辑命名的模型，以改变关系的开始和结束节点。让我们生成开始于节点 c 的 B2A 关系：

```
MATCH (a:FIRST:MERGE)-[rel:B2A]-(b:SECOND:MERGE), (c:THIRD)
CALL apoc.refactor.from(rel,c)
```

我们获得如图 12-2 所示的展示图。

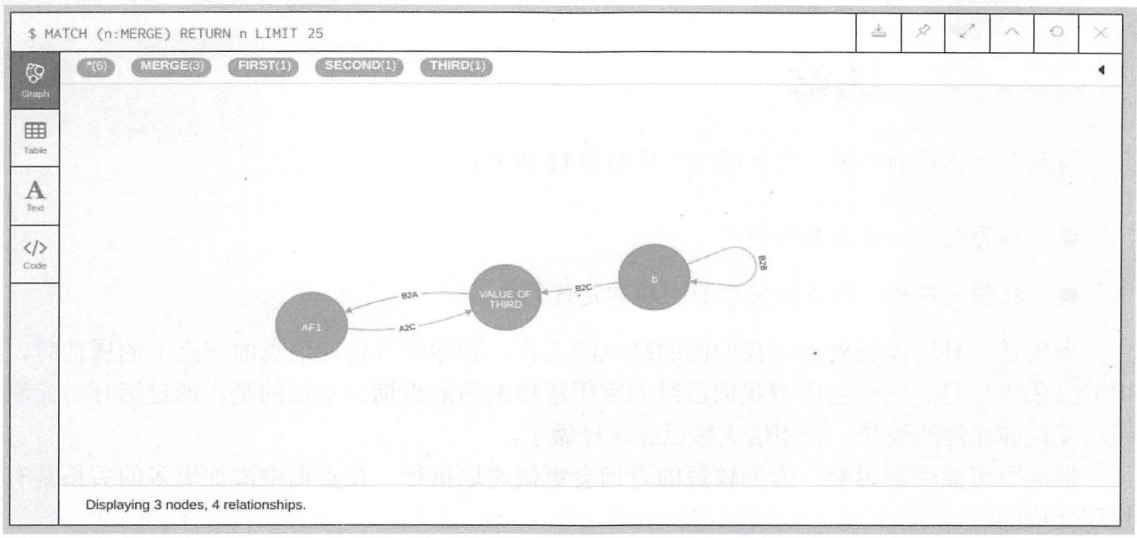


图 12-2　从 c 开始的 B2A 关系

现在让我们生成结束于节点 a 的 B2B 关系：

```
MATCH ()-[rel:B2B]-(), (a:FIRST)
CALL apoc.refactor.to(rel, a) YIELD output
RETURN output
```

这戏剧性地改变了我们的图，有些非常不合逻辑，如图 12-3 所示。

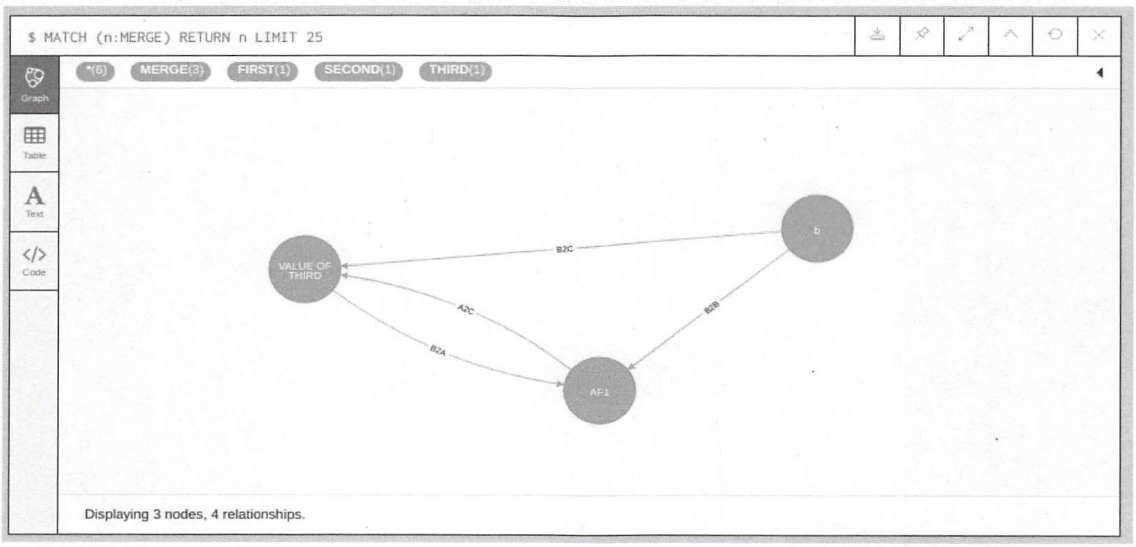


图 12-3　发现谎言关系

注意，这个结果是有效的，因为只有一个节点对应一个标签——First、Second 以及 Third。如果我们注意到仅对标签名使用大写字母的错误，应额外奖励。

12.4 结论

有两个我喜欢的谚语，完全适用于我们的 IT 世界：

- 权力越大，责任越大……
- 犯错是人类，大多数灾难影响的却是计算机。

重构是一种可能意外破坏我们图的高风险工作。删除没有转换信息的节点上的属性时，可能会丢失信息。它还会影响我们已经为应用程序编写的查询。幸运的是，通过运行单元测试会发现破坏性的改变，我相信大家已经这样做了。

性能也可能受到影响，因为较短的查询会更快速地执行。在查询中添加更多的关系是有性能代价的。

12.5 小结

我们看到了如何修改在最常见更新场景下的数据图。毫无疑问，这对在生命周期内（可能它们的寿命长，且生命旺盛）处理数据项目的演变是有用的。

在下一章中，我们将讨论若干 Neo4j 企业服务器组成的集群。这是新的知识！

第 13 章

◂集　群▸

本章将介绍 Neo4j 3.2 企业版的集群功能，特别是 Neo4j 的因果集群（causal clustering）。

集群就是指多个服务器共同协作完成任务，它们的特性用一句谚语来描述就是“United we stand, divided we fall and E pluribus unum（我们身为一体又相互独立，我们合众为一）”。为了更方便阐述集群的特性，我们将使用两台 Linux 笔记本电脑和几台树莓派设备（Raspberry Pi，一款由英国“The Raspberry Pi Foundation”慈善组织针对计算机业余爱好者、教师、小学生以及小型企业等用户推出的迷你计算机，通常预装 Linux 系统，体积仅信用卡大小，搭载 ARM 架构处理器，运算性能和智能手机相仿）来做实验。

由于集群是企业版特有的功能，因此它在社区版本上是没有的。

本章包括以下内容：

- 为什么需要 Neo4j 集群
- Neo4j 集群的概念
- 创建 Neo4j 集群
- 在 Neo4j 集群中实现灾难恢复

13.1 为什么需要 Neo4j 集群

建立集群的主要原因是有时候单个服务器不足以负担起整个业务。如果业务非常庞大，我们就可以在世界上不同的地区设立多个服务器，使它们组成一个集群，让这些服务器共同协作完成整个业务，这样问题就得以解决了。所以集群是解决这类业务的最佳方案。使用集群有三个主要的优势：

- 高吞吐量。
- 数据冗余。
- 高可用性（HA）。

如果业务需要 7×24 的高可用性，那就可以考虑采用集群了。

13.2 Neo4j 集群的概念

在 Neo4j 因果集群中有两类服务器：核心服务器和只读副本服务器。其中，核心服务器是最重要的，只读副本服务器相对次要，因为只读副本服务器可以在扩展集群的过程中任意添加或删除。让我们来看看两者各自所负责的角色。

13.2.1 核心服务器

核心服务器是集群中最重要的服务器。在运行一个事务时，当在一个 Neo4j 集群中拥有至少 3 个核心服务器的情况下，只要事务已经被总服务器数量的一半加一台服务器成功执行，那么集群就可以向用户报告事务执行成功。

Neo4j 集群所担负的角色：

- 它们是所有更新操作的目标服务器，并通过一种共识机制来维护数据的一致性。
- 它们负责数据安全。
- 它们使用 bolt 协议完成查询路由。
- 它们将数据复制到只读副本。

在一个 Neo4j 集群中，只有一台服务器是领导者（leader），并由其他核心服务器选出。这个机制对用户来说是透明的。

13.2.2 只读副本服务器

只读副本服务器就是用来完成**只读功能的服务器**。它们的数据由核心服务器来更新。在事务中，如果事务还没有提交成功那么只读副本服务器的数据在某个时间点可能不是最新的。只读副本服务器一般是用作大量查询操作的，如生成报表。

13.2.3 高吞吐量

当你将服务器投入使用时，使用用户越多，服务器的功耗就越大。

目前主流的网站当用户增加时并不会去购买更大型的服务器，而是采用负载均衡的解决方案。负载均衡的概念就是：整个服务集群有一个入口点负责接收请求，接收到请求后就将任务发送给下一级，并且下一级就是由众多其他的负载均衡服务器组成的，这些负载均衡服务器可能是硬件或软件（如 NGINX）。

Neo4j 的新协议可以实现负载平衡。因此，包含众多只读副本服务器的 Neo4j 集群可以实现更高的吞吐量，相当于 Neo4j 集群组成为一台更强大的机器。查询可以分开执行，以便在多台机器上各自并行执行。

13.2.4　数据冗余

数据冗余是防止数据丢失的解决方案。数据以异步的方式自动从一个节点复制到另一个节点。

当用户更新数据时，更新操作将从多个核心服务器上分派到剩余的核心服务器和只读副本服务器。通常不会将所有服务器都放在同一个房间里，以防止特定灾难的降临。

13.2.5　高可用

如果你是服务提供商，你的合同中肯定会有一条关于 99%可用性的条款。高可用性是通过因果集群体系架构实现的。如果服务器发生故障，当某台服务器发生严重故障，则其他在线服务器将自动处理这些请求。

13.2.6　Bolt 协议

Bolt 是一个用于与 Neo4j 服务器进行通信的二进制协议。它有三个主要特征：轻量、高效和类型安全（如发送'1'字符，获取到的也是字符，而不是整数）。这个协议的一个变体是 bolt + routing，从版本 3.1 开始，负载平衡已集成到驱动程序和 Bolt 协议中。

13.3　构建 Neo4j 集群

本节我们将使用到的硬件设备有：

- 两台笔记本电脑作为核心服务器，运行 Linux，连接到家庭 WiFi 网络。
- 两个树莓派二代或三代迷你计算机，作为核心服务器运行。

只读副本服务器将在 Docker 容器中运行。我的主笔记本电脑上将运行五个副本。

在典型的部署案例中，通常只读副本服务器的数量比核心服务器多。当然，只要可以连接到网络并运行 Neo4j，就可以使用任何你可用的机器，包括 PC、Mac 或运行有 Linux 的任何品牌的信用卡大小的计算机。

在家庭环境中，所有服务器都在同一个网络上，插在同一个路由器上，并且大部分在同一个房间（因为 Wi-fi 和电缆原因）。但是，这种网络环境仍然可以用来运行我们的实例，并且这也是使用硬件构建集群案例最便宜的选择之一。

在商业环境中，可能须与服务器提供商洽谈：声明使用几台机器、列出使用的端口，并要求网络数据包能够安全传输。这可能需要很长时间。

13.3.1　核心服务器

由于树莓派是在没有人为干预的情况下运行，因此没有添加配置其 IP 地址，而是依靠我的路由器的 DHCP 自动分配其 IP 地址。如果你也是同样的情况，你需要知道你的设备在重

启（或者 DHCP 时间到期）时可能会得到一个不同的 IP 地址。

硬件准备完毕后，下一步我们查看网络上有多少个主机；我为此写了一个简短的 shell 脚本，你可以在附录的提示和技巧中找到这个脚本，它可以计数并返回所有 IP。在当前情况下，运行该脚本，路由器下的所有设备都会被一一列出（例如智能手机、电视机甚至冰箱）。

然后是准备软件部分。首先，我建议你对所有设备都更改默认密码，当然我们可以将所有的树莓派设备都更换为相同的新密码，以方便远程连接。并且，笔记本电脑要确保能够通过 SSH 登录到其他设备，具体见附录的提示和技巧。如果无法连接到树莓派，请在树莓派上启用 SSH。

其次是安装最新版的 Java（sudo aptitude install openjdk-8-jdk）。

最后，也是最重要的：安装 Neo4j（像往常一样我们采用解压版的 Neo4j）。

要将 Neo4j 程序从主笔记本电脑复制到另一台运行 SSH 的主机，请按如下方式使用 scp 命令：

```
scp neo4j.zip pi@192.168.0.20:/home/pi/
```

在此，我建议在其中一台主机上安装 APOC 插件。

为了检验每台设备上 Neo4j 的可用性，我们可以在每台机器上启动 Neo4j 来检查它是否能成功启动并运行。然后停止它们并删除$NEO_HOME/data/databases 的内容。现在，列出你的服务器，并设置哪些是核心服务器、哪些是只读副本。在我的情况下，核心服务器是那些 IP 地址以 12、15 和 16 结尾的机器（11 和 20 作为只读副本）。可能有 5 个或 20 个服务器，我们需要配置核心服务器和只读副本服务器。

对于服务器，请将以下内容添加到$NEO_HOME/conf/neo4j.conf 文件的末尾：

```
dbms.mode=CORE
causal_clustering.expected_core_cluster_size=4
causal_clustering.initial_discovery_members=192.168.0.12:5000,192.168
.0.15:
5000,192.168.0.11:5000,192.168.0.20:5000
causal_clustering.discovery_listen_address=HostsIPAddress:5000
causal_clustering.transaction_listen_address=HostsIPAddress:6000
causal_clustering.raft_listen_address=HostsIPAddress:7000
dbms.connectors.default_advertised_address=HostsIPAddress
```

接下来我们需要修改每台主机上的配置文件，将 HostsIPAddress 替换为 IP 地址。我们希望能够从另一台主机打开它们的 Web 界面。因此，在 neo4j.conf 中将以下行前面的#号注释符去掉：

```
dbms.connectors.default_listen_address=0.0.0.0
```

如果我们有更多的硬件来配置为只读副本服务器，则除了第一行外它们的配置大致相同：

```
dbms.mode=READ_REPLICA
causal_clustering.expected_core_cluster_size=4
causal_clustering.initial_discovery_members=192.168.0.12:5000,192.168
.0.15:
```

```
5000,192.168.0.11:5000,192.168.0.20:5000
```

也可以将 Neo4j 托管在某个云平台上并创建因果集群。如果你想建立这样一个集群，你需要联系 Neo4j 的销售代表来处理此事。

接下来，让我们启动 Neo4j：

```
neo4j start
```

如果某个服务器启动 Neo4j 需要很长的时间。我们可以通过查看日志文件来追踪进度：

```
tail -f neo4j.log
```

当日志中最后一行为：

```
INFO Remote interface available at http://192.168.0.12:7474/
```

我们就可以使用浏览器打开在 Neo4j 的主页了。你需要登录 Neo4j 并为默认用户设置一个新密码。只有这样你才能在第一个选项卡（堆栈图标）的左栏中看到主机的角色。最开始启动的将是 LEADER，而其他的将是 FOLLOWER。

现在，我们来检验数据是否可以在集群中复制。最简单的方法是打开 LEADER 核心服务器主机的浏览器并创建一些数据，然后打开另一台主机的浏览器并查询该数据：

```
CREATE (:Person:RoleModel {name:'Didier'}), (:Person:Sun {name:'Anny'}),
(:Person:Mum {name:'Josiane'}), (:Person:Dad {name:'Jean-Pierre'})
```

运行上面的语句后，其他主机上也应该能够查询到相同的数据了。

如果你尝试在 FOLLOWER 上创建节点，无论是使用 Neo4j 浏览器还是使用 Cypher shell，都会收到错误消息。我在稍后介绍为什么会这样。

13.3.2　只读副本服务器

如前所述，我们将使用 Docker 容器来创建只读副本服务器。为确保数据得到复制，我们将使用本地文件夹来存储每个副本的日志和数据文件。这不是强制性的，但会帮助我们确保集群的成功运行。

我将副本命名为：a、b、c、d、e 和 f。在我的 home 文件夹中，我将为每个主机创建专用文件夹：ReadReplicas 和 subtree。

在 Linux 机器上，完成以上操作的 shell 命令如下所示：

```
cd ~
mkdir ReadReplicas
# Repeat lines below for each host
mkdir ReadReplicas/a/
mkdir ReadReplicas/a/logs
mkdir ReadReplicas/a/data
```

在此没有 plugin 文件夹也没有 conf 文件夹。为了确保我们想要的配置生效，需要在

neo4j.conf 配置文件中额外设置一些参数，这些参数的格式也需要转变下：

```
causal_clustering.expected_core_cluster_size=4
 转变为：
 NEO4J_causalClustering_expectedCoreClusterSize=4
```

请注意上面的 NEO4J_前缀、下画线和骆驼风格的大小写。

现在，要启动一个只读副本服务器，Docker 命令如下所示：

```
docker run --name=replica-a --detach --
env=NEO4J_dbms_mode=READ_REPLICA \
--env=NEO4J_causalClustering_expectedCoreClusterSize=4 \
--
env=NEO4J_causalClustering_initialDiscoveryMembers=192.168.0.12:5000,
192.16
8.0.15:5000,\
192.168.0.11:5000,192.168.0.20:5000 \
--volume=$HOME/ReadReplicas/a/logs:/logs \
--volume=$HOME/ReadReplicas/a/data:/data neo4j:3.2.3-enterprise
```

我们按照相应的主机名称来运行上面的命令后，可在 ReadReplicas 文件夹中运行以下命令以查看文件夹的大小，随着副本从核心服务器获取数据，文件夹就会变大：

```
du -ch |grep total
```

注意，如果 7474 端口没有被设置为公开，那么在其他设备上是无法使用浏览器访问它的 Neo4j 页面的。

13.3.3 bolt+routing 协议

bolt+routing 协议是 bolt 协议的变体。默认情况下 bolt 是默认协议，但可以在客户端代码中选择使用 bolt + routing 协议。这将允许我们指定 FOLLOWER 核心服务器作为数据更新的接收者，因为服务器和驱动程序会将更新的数据通过路由传送到 LEADER。

如果 LEADER 服务器宕机，那么 FOLLER 服务器就有可能被选举为 LEADER 服务器。如果这种情况发生了，大多数情况下，核心服务器将进入一个选举过程。

为了能够在 FOLLOWER 主机的 Neo4j 浏览器中运行更新语句，我们可以点击带齿轮图标的选项卡并在左栏中勾选 bolt+routing。然后，如下所示的更新语句将正确运行：

```
CREATE(ju:Person {name:"MonChéri"}), (am:Person {name:"MaPrincesse"})
```

作为一个练习，你可以尝试停止 LEADER(neo4j stop)，并观察哪一个主机将变成 LEADER。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

13.4 在 Neo4j 集群中实现灾难恢复

有时服务器会因为某些原因发生故障，如果一个只读副本服务器发生了故障，那么集群中其他的或者一个新加入集群的只读副本服务器就会替换它。它一旦加入集群之后，就会获取数据。如果是一个核心服务器发生故障，我们也会有办法处理它：图数据会通过其他核心服务器重新创建。步骤为：

步骤 01　停止一个核心服务器。
步骤 02　浏览其文件夹，删除 data/databases 和 data/cluster-state 文件夹内容。
步骤 03　重新启动核心服务器（数据越多，需要的时间越多）。
步骤 04　查看日志文件以确定它何时准备就绪。
步骤 05　用浏览器打开它的 Neo4j 主页。
步骤 06　完成。

这很简单，它让我想起了一位演讲者在一次图技术会议上所说的话：“我们把困难的事情都做了，所以你不必再这样做。”

如果所有核心服务器都发生故障，请首先查找原因，因为这种情况在真正的最新的部署中不太可能发生。如果要将图数据回滚到以前的某个备份，步骤如下所示：

步骤 01　停止所有实例（核心和副本）。
步骤 02　对每个实例执行备份还原（使用相同的备份文件）。
步骤 03　重新启动所有实例。

在回滚过程中集群是无法对外提供服务的。

13.5 小结

本章告诉我们为什么以及如何使用不同种类的服务器创建 Neo4j 集群。在介绍完集群的概念之后，我们看到了如何构建一个集群，也认识了 bolt+routing 协议的魔力。

你了解到，我们可以从树莓派（https://www.raspberrypi.org/）等几个信用卡大小的计算机集群发展成为由多个云平台服务器组成的集群（你需要一张强大的信用卡）。

我们还看到了一个灾难恢复过程，另外我给出以下提示：

无论使用任何软件，灾难恢复必须始终是团队合作。否则，最终责任将会被归咎于某一个人身上。

如果你是善于了解失败的人并且是知道如何解决问题的人，那么就应该承担责任。我亲眼看到一位客户因为厌倦了这点而失去了最好的系统管理员。

接下来了解一个 Neo4j 示例：影响分析和模拟。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

第 14 章

◀ 案例——推荐系统 ▶

在本章中，我们将讨论如何更好地把 Neo4j 图数据库应用到具体案例中。前文所有的讨论都是为本章做铺垫，下面我们开始一起探讨如何把 Neo4j 这样的图数据库实际应用到具体案例中。

我们将在本章讨论以下内容：

- 商品关系图建模
- 推荐系统关系应用
- 欺诈检测关系应用

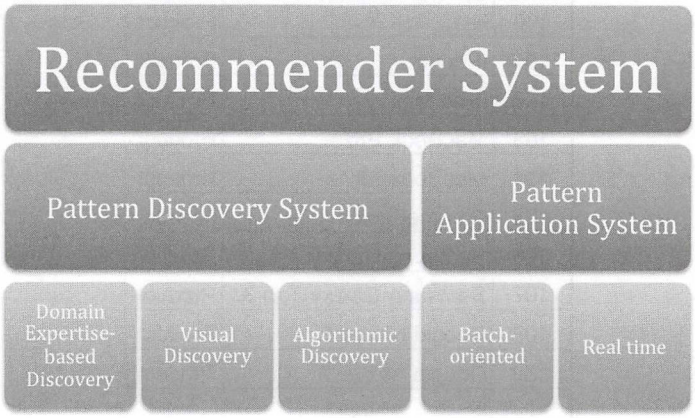
14.1 推荐系统剖析

我们将在本章中讨论大家熟知的推荐系统（见图 4-1）。这类系统通常包括两个基本部分：

- **模式识别系统**：对于特定的目标群体，系统会以某种方式计算出有用的推荐。这种识别可以用很多不同的方法来完成，但总体来说，可以分为以下三种方法：
  - **业务专家**能够深入了解图数据库应用领域，并使用这种方法获取有用的推荐。例如，一家 DIY 零售店的主管会明白一个特定的场景，假设有人来买多罐涂料，那么他们可能也会对高端画笔的推荐感兴趣。如果商店正好有此类促销活动，那么推荐就会很及时。这就是一个模式识别过程，业务专家可以通过图数据库实时识别这种模式，并作为复杂推荐的一部分。
  - **视觉发现**是一种在业务领域使用图展示的特定模式。我们已经发现在许多不同的项目中，商业用户在使用图可视化来查看他们的数据时偶然发现了这些类型的模式。出现特定模式，跳出意外关系，或者在更先进的可视化解决方案中，特定的集群活动突然变得清晰可见并需要做进一步调查。图数据库（如 Neo4j）和可视化方案，在这个过程中可以发挥非常强大的作用。
  - **算法识别**是使用机器学习算法在业务领域的数据集中揭示数据中以前未知的模式。通常，这些过程需要一个迭代的、原始的数字处理方法，在过去的非图数

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

据格式中也使用过这种方法。在这一点上，它部分是艺术，部分是科学，但如果应用正确，能产生很多有趣的洞察。



推荐系统简介

- **模式应用系统**：所有的推荐系统能获得或多或少的成功，不仅需要去发现这些模式，还需要能够把它们应用到实际的业务中去。我们下面会介绍基于这些模式的不同类型的应用程序：
  - **面向批处理的应用程序**：这些模式的应用程序并不像我们所期望的那样具有时间性。例如，大量的商品推荐甚至优惠券等电子邮件在上午 10 点或 11 点发送给客户（或潜在客户）那里。批处理方案通常可以处理这类请求，虽然这类推荐模式看起来最低效。
  - **实时应用程序**：这些模式应用程序必须实时交付，在网页请求和响应之间，不能预先计算。对于这些类型的系统，通常我们要使用更复杂的数据库查询来匹配适当的推荐，像 Neo4j 这样的图数据库是一个很好的工具。我们将进一步说明这一点。

有了这些分类，我们通过一些数据集和查询实例来更形象地阐述推荐系统。

14.2 推荐系统的图模型应用

我们将在推荐系统中使用一种特别的数据模型。假设我们一共有这些数据：

- 10 种商品
- 3 种品牌
- 顾客节点和商品之间存在的 50 个关系，这些关系能够记录顾客购买了哪些商品。

表 14-1 是我们添加的商品和品牌。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 14-1　添加商品和品牌到数据集

Node	Name	Label
100	iPad	Product
101	iPhone	Product
102	iPad Mini	Product
103	MacBook Pro	Product
104	MacBook Air	Product
105	ChromeBook	Product
106	Samsung Galaxy 4	Product
107	Samsung Galaxy Tab 3	Product
108	Google Nexus 5	Product
109	Google Nexus 7	Product
⋮	⋮	⋮
200	Apple	Brand
201	Google	Brand
202	Samsung	Brand

图 14-2 显示了结果模型。

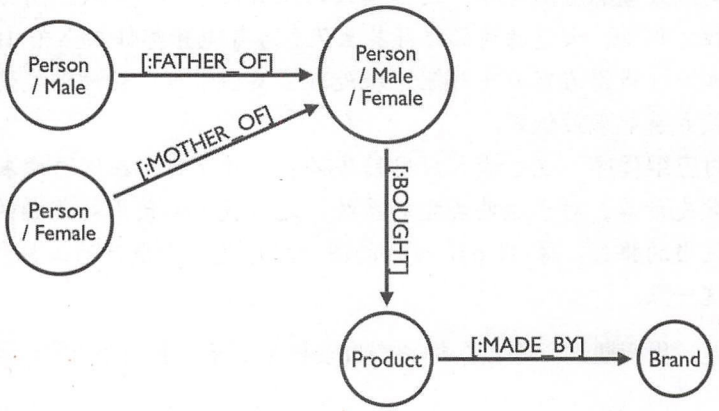


图 14-2　元模型

在 Neo4j 中，这个模型看起来如图 14-3 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

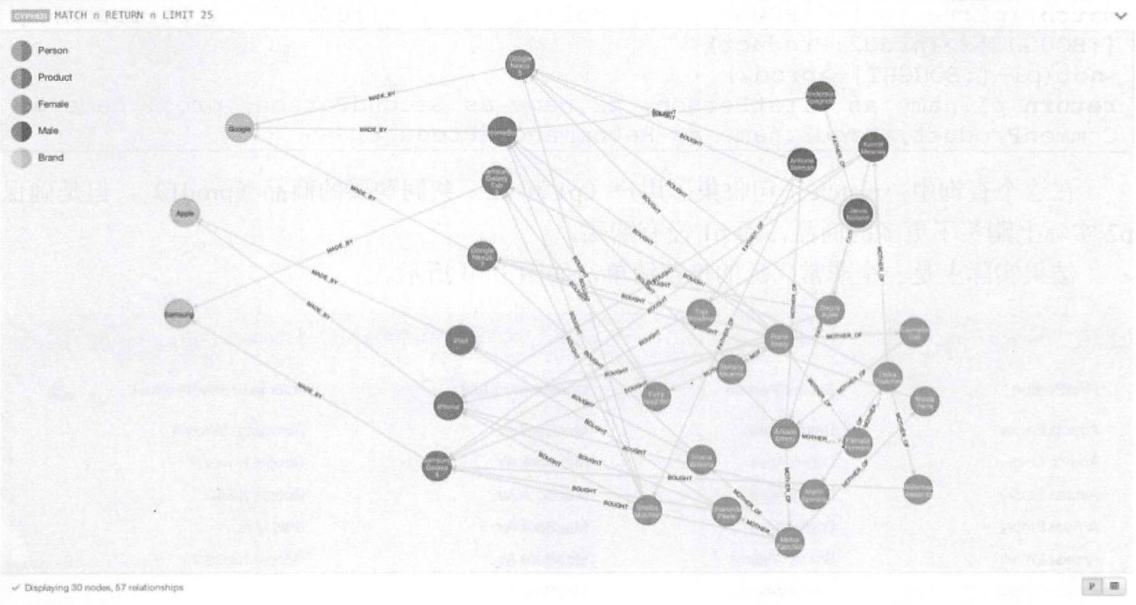


图 14-3　数据集视图

虽然这是一个很简单的数据集，但是却为我们的推荐系统提供了一些有趣的尝试。下面，我们将在商品推荐系统中通过一些查询来可视化或实时地挖掘出一些数据。

14.3 推荐系统查询示例

在示例数据集中，我们将探讨几个有趣的查询，这些查询将允许我们使用一些可用的信息来为我们假想的用户构造有趣的建议。我们将沿着以下几个不同的维度进行：

- 商品购买
- 品牌忠诚度
- 社会或家庭关系

让我们先从商品购买开始。

14.3.1 基于商品购买的推荐

让我们从头开始考虑。我们要编写的第一个查询是基于过去的购买行为。我们希望找到那些已经分享了他们过去购买过的商品的人，但这也很显然找到了不分享的商品。在我们的数据模型中，这个 Cypher 查询的内容如下：

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
match (p1:Person)-[:BOUGHT]->(prod1:Product)<-[:BOUGHT]-(p2:Person)-
[:BOUGHT]->(prod2:Product)
 not(p1-[:BOUGHT]->prod2)
return p1.name as FirstPerson, p2.name as SecondPerson, prod1.name as
CommonProduct, prod2.name as RecommendedProduct;
```

在这个查询中，match 子句收集了用户（p1 和 p2）共同购买的商品（prod1），但是确保 p2 实际上购买了更多的商品，而 p1 没有购买。

结果实际上是一个非常广泛的推荐清单，如图 14-4 所示。

CYPHER match (p1:Person)-[:BOUGHT]->(prod1:Product)<-[:BOUGHT]-(p2:Person)-[:BOUGHT]->(prod2

FirstPerson	SecondPerson	CommonProduct	RecommendedProduct
Amada Emory	Dagny Agee	MacBook Air	Samsung Galaxy 4
Amada Emory	Dagny Agee	MacBook Air	Google Nexus 5
Amada Emory	Dagny Agee	MacBook Air	Google Nexus 7
Amada Emory	Dagny Agee	MacBook Air	iPad Mini
Amada Emory	Shana Willems	MacBook Air	Google Nexus 7
Amada Emory	Shana Willems	MacBook Air	iPad
Amada Emory	Antione Selman	MacBook Air	Google Nexus 5
Amada Emory	Antione Selman	MacBook Air	MacBook Pro
Amada Emory	Rana Seely	MacBook Air	iPhone
Amada Emory	Rana Seely	MacBook Air	Google Nexus 5
Amada Emory	Rana Seely	MacBook Air	Google Nexus 7
Amada Emory	Fairy Daughtery	ChromeBook	Samsung Galaxy Tab 3

Returned 249 rows in 524 ms

图 14-4　广泛的推荐清单

所以我们需要做一些改进。

下一步我们要看这两个人是否有更强的相似性，譬如两个人购买两个以上共同的商品，那么推荐的商品会对目标受众更有吸引力。

这个查询是：

```
match (p1:Person)-[:BOUGHT]->(prod1:Product)<-[:BOUGHT]-(p2:Person)-
[:BOUGHT]->(prod2:Product)
with p1,p2,count(prod1) as NrOfSharedProducts, collect(prod1) as
SharedProducts,prod2
where not(p1-[:BOUGHT]->prod2) AND NrOfSharedProducts > 2
return p1.name as FirstPerson, p2.name as SecondPerson, extract(x in
SharedProducts | x.name) as SharedProducts, prod2.name as
RecommendedProduct;
```

正如我们看到的，基本的查询与上一个相同，但我们为共同拥有已购商品数量添加了一些过滤条件。我们对共同拥有的商品集合（**collect(prod1)**）进行处理并提取名称作为最终结果。结果如图 14-5 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
CYPHER match (p1:Person)-[:BOUGHT]->(prod1:Product)<-[:BOUGHT]-(p2:Person)-[:BOUGHT]->(prod2:Product)
```

FirstPerson	SecondPerson	SharedProducts	RecommendedProduct
Dagny Agee	Rana Seely	MacBook Air, Google Nexus 5, Google Nexus 7	ChromeBook
Shelba Mutchler	Rana Seely	iPhone, ChromeBook, Google Nexus 5	Google Nexus 7
Dagny Agee	Rana Seely	MacBook Air, Google Nexus 5, Google Nexus 7	iPhone
Rana Seely	Dagny Agee	MacBook Air, Google Nexus 5, Google Nexus 7	Samsung Galaxy 4
Rana Seely	Shelba Mutchler	ChromeBook, iPhone, Google Nexus 5	Samsung Galaxy Tab 3
Shelba Mutchler	Rana Seely	iPhone, ChromeBook, Google Nexus 5	MacBook Air
Rana Seely	Dagny Agee	MacBook Air, Google Nexus 5, Google Nexus 7	iPad Mini

✓ Returned 7 rows in 192 ms

图 14-5　基于商品购买的改进推荐

现在我们来看看另一种基于品牌忠诚度的推荐。

14.3.2　基于品牌忠诚度的推荐

众所周知，如果我们已经拥有某品牌的商品，很可能会对同一品牌的其他商品也感兴趣。所以，让我们找到那些已经购买过某品牌商品的用户，看看是否可以推荐该品牌的其他商品。以下是这个查询示例：

```
MATCH (p:Person)-[b:BOUGHT]->(prod1:Product)-[:MADE_BY]->(br:Brand)<-
[MADE_BY]-(prod2:Product)
WITH p, br, prod2, count(prod1) as NrOfBrandProducts
WHERE not(p-[:BOUGHT]->prod2) and NrOfBrandProducts > 1
RETURN p.name as Person, br.name as Brand, collect(prod2.name) as
RecommendedProducts
ORDER BY Person ASC;
```

这种模式与之前相似，不同之处在于，我们现在正在计算某品牌的商品数量，并确保该用户没有购买同一品牌的其他商品。结果如图 14-6 所示。

```
CYPHER match (p:Person)-[b:BOUGHT]->(prod1:Product)-[:MADE_BY]->(br:Brand)<-[MADE_BY]-(prod2:Product)
```

Person	Brand	RecommendedProducts
Antione Selman	Apple	iPad Mini, iPhone, iPad
Dagny Agee	Google	ChromeBook
Dagny Agee	Apple	iPhone, iPad, MacBook Pro
Rana Seely	Apple	iPad, MacBook Pro, iPad Mini
Shana Willems	Apple	iPhone, MacBook Pro, iPad Mini
Sharonda Peele	Apple	MacBook Pro, iPad Mini, MacBook Air
Shelba Mutchler	Google	Google Nexus 7

✓ Returned 7 rows in 104 ms

图 14-6　基于品牌的推荐

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

给予不同的限定条件，我们获得了一些非常有用的推荐。接下来让我们引入另一个参数：社交关系。在这个例子中，我们将基于特定用户的家庭关系来做一些推荐。

14.3.3 基于社交关系的推荐

在接下来的推荐查询中，我们将使用社交关系。在特定的数据集中，我们将基于父母、兄弟等关系来为用户下一次购买提出有用的推荐。

让我们看看下面的查询：

```
match (p:Person)-[b:BOUGHT]->(prod:Product),p<-[r1]-(parent:Person)-
[r2]->(sibling:Person)
where type(r1) in ["MOTHER_OF","FATHER_OF"] and type(r2) in
["MOTHER_OF","FATHER_OF"]
and not(sibling-[:BOUGHT]->prod)
return p.name as Person, prod.name as RecommendedProduct,
collect(sibling.name) as ForSiblings;
```

以上查询将找出购买过某个商品的用户，并寻找没有购买这些商品并可能从中受益的兄弟或姐妹（有相同的母亲或父亲）。运行该查询将得到如图 14-7 所示的结果集。

CYPHER match (p:Person)-[b:BOUGHT]->(prod:Product), p<-[r1]-(parent:Person)-[r2]->(sibling:F

Person	RecommendedProduct	ForSiblings
Pamala Forward	iPad Mini	Antione Selman, Melva Fairchild
Antione Selman	Google Nexus 5	Melva Fairchild, Pamala Forward
Shelba Mutchler	Samsung Galaxy Tab 3	Tisa Woodman
Shelba Mutchler	Google Nexus 5	Tisa Woodman
Shelba Mutchler	iPhone	Tisa Woodman
Kermit Meaney	iPhone	Williemae Dossantos
Williemae Dossantos	Samsung Galaxy 4	Marth Sparling
Shelba Mutchler	ChromeBook	Tisa Woodman
Kermit Meaney	Samsung Galaxy 4	Marth Sparling
Melva Fairchild	MacBook Pro	Pamala Forward
Antione Selman	MacBook Air	Melva Fairchild, Pamala Forward
Melva Fairchild	Samsung Galaxy 4	Antione Selman, Pamala Forward

Returned 32 rows in 119 ms

图 14-7 基于社交关系的推荐

同样，这是有用的推荐。现在，让我们试着把这一切结合起来，并尝试生成基于所有这些标准的复杂的实时推荐：商品购买历史、品牌忠诚度以及社交网络。

14.3.4 组合推荐

在下面的查询中，我们将尝试将前面提到的三个维度组合在一起，以提供更加有关联性的推荐。如果我们把所有这三个角度都考虑到推荐系统中，那么我们就能给用户提供更精准的推荐。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

让我们看看下面的查询：

```
match (p1:Person)-[:BOUGHT]->(prod1:Product)<-[:BOUGHT]-(p2:Person)
[:BOUGHT]->(prod2:Product), p1<-[r1]-(parent:Person)-[r2]->p2, prod1
[:MADE_BY]->(br:Brand)<-[:MADE_BY]-(prod2)
where type(r1) in ["MOTHER_OF","FATHER_OF"] and type(r2) in
["MOTHER_OF","FATHER_OF"] and not(p1-[:BOUGHT]->prod2)
return p1.name as FirstPerson, p2.name as SecondPerson, br.name as
Brand,
prod2.name as RecommendedProduct;
```

这里我们所做的是将三个维度的模式放在 match 子句中。以下是不同部分介绍：

（1）第一部分确保两个人购买了相同的和不同的商品。

（2）第二部分确保这两个人是兄弟姐妹。

（3）第三部分确保推荐的商品是基于对某一特定品牌的忠诚度。

运行这个查询很有趣。结果如图 14-8 所示。

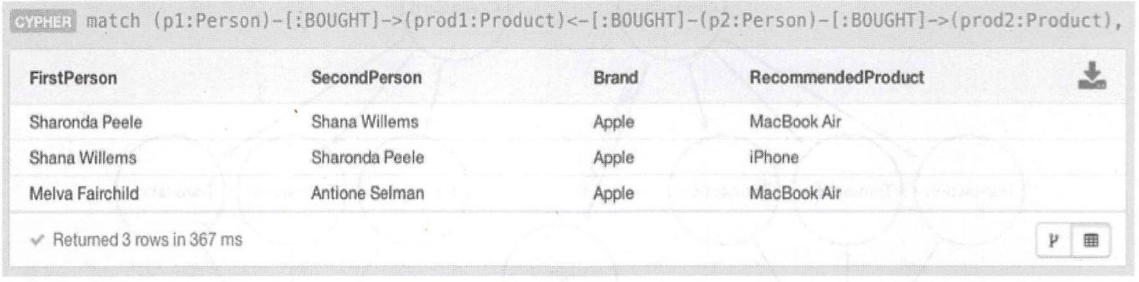


图 14-8　组合推荐

我们很快看到，这个复杂的模式只得到了 3 个匹配的结果，然而，正如前面提到的，我们有理由相信这些推荐将会更加有力。

关于推荐系统，我们核心案例的讨论到此为止，在结束本章之前，我们再花点时间来讨论关于推荐系统的一些相关话题。

14.4 推荐系统的业务变化

正如我们之前所描述的，推荐系统的整个原理可以推广到其他商业应用，生成不同类型的系统。有些人称之为规则引擎，通过一系列复杂的 if-this-then-that 匹配，算出在决策树的另一端该采取什么行动。另一些人称之为模式匹配系统，它可以应用于任何类型的模式，并与任何类型的行为联系在一起。最有可能的是，像 Neo4j 这样的图数据库具有上述所有特性，并为我们提供了一个有趣的基础结构优化，可以很好地服务于这些系统。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

在结束本章之前，我们想着重介绍一些与推荐系统极其相关的案例。让我们来看一些众所周知的最佳应用程序，它们本质上使用了相同的原理。

14.5 欺诈检测系统

我们已经看到许多客户使用图数据库管理系统（比如 Neo4j）来用于欺诈检测系统。原理很简单：在许多情况下，特定的欺诈行为不仅仅只是通过一次交易发，它需要通过一连串的交易来分析是否有一些具体的特，通过相互对比来看看是否真的构成欺诈。

在下面的例子中，我们来看看信用卡欺诈的一个可疑案例（见图 14-9）。

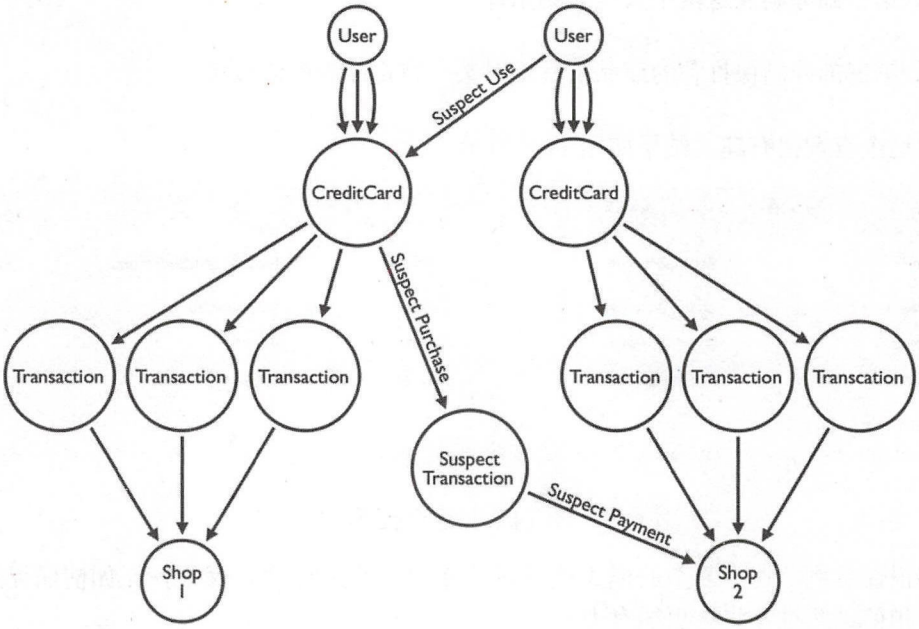


图 14-9　信用卡欺诈疑似案例

通常来说，一个特定的用户总是使用他的信用卡在特定的商店进行交易，而另一个用户会使用他的信用卡在另一个特定的商店进行类似的交易。如果中间突然有了新的交易，用户在另一家商店中使用信用卡进行了类似的交易，那么在某些欺诈检测系统中就可能会被标记为可疑模式。

系统并不一定会立即阻止信用卡交易，但相关的信用卡交易组合的风险得分肯定会上升。如果分数达到了某一临界值，就意味着该交易存在欺诈的可能性增加，系统会建议采取特定的行动。

这个行动并不是去推荐商品的购买，而是将信用卡暂停服务，并给用户一个友好的客户服务电话以检查是否为异常交易。然而，这种欺诈检测系统的原理与零售推荐系统非常相似：定义模式，检测模式，并采取某种与业务相关的措施来处理该模式的出现。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

14.6 访问控制系统

另一个相似的例子是访问控制系统（见图 14-10），它采用了类似的推荐系统原理——定义模式，然后匹配它的出现。

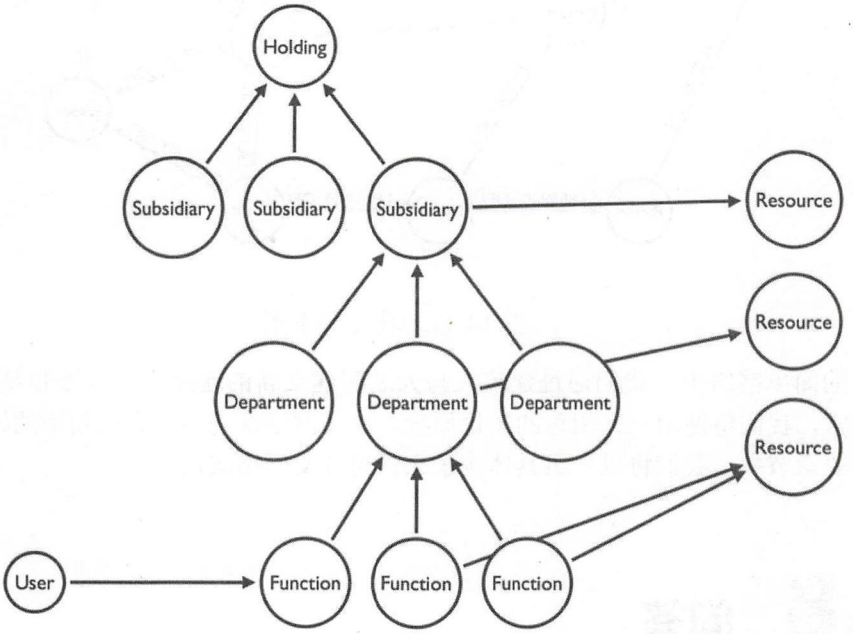


图 14-10　访问图

14.7 社交网络系统

很明显，有很多推荐系统都是专门针对某个领域的。在过去的几年里，随着社交网络工具和社交应用的大量涌现，人们对社交推荐系统的兴趣大增。从本质上讲，我们希望帮助用户之间建立起新的连接，虽然用户可能没有意识到，但却可以帮助在同一个社交圈提升社交效率。

为清楚地阐述上述观点，请查看图 14-11 中的示例。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

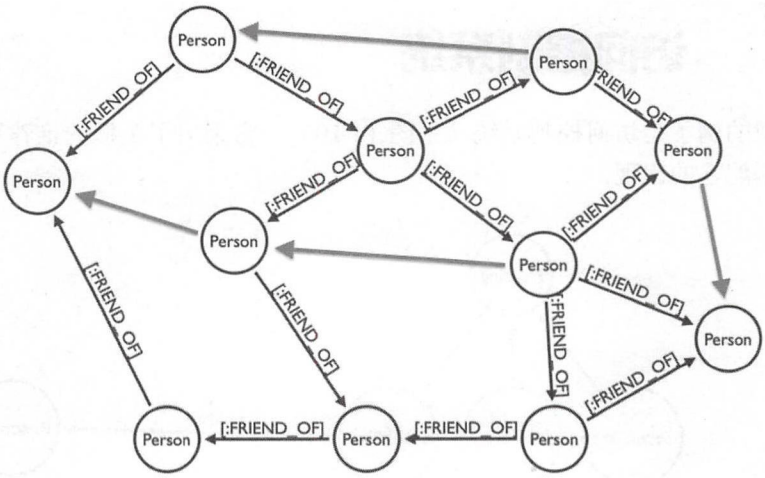


图 14-11　社交关系图

在前面的简单网络中，我们通过建议人与人之间建立新的连接，可以很容易地建立朋友的闭环。通常，我们将使用三元闭包的图论原理，以在网络结构中找到三角形的缺失环节。

在后续的章节中，我们将以一组具体例子来探讨社交网络案例。

14.8 问答

问题 1：为了创建一个推荐系统，需要人工智能引擎来查看我的数据并自动发现推荐模式。

A. 是

B. 否

正确答案：**否**。推荐系统可以基于你已经拥有的业务知识，在浏览数据时发现推荐的可视化模式，或基于某种机器学习算法。这三种方法都可以为你的业务应用程序提供有意义的推荐模式。

问题 2：推荐系统只适用于亚马逊这种类型的零售环境，在那里你拥有大量的数据来帮你推荐的依据。

A. 是

B. 否

正确答案：**否**。推荐系统在许多不同的业务领域都很有用，不仅仅是零售商品推荐。欺诈检测系统（推荐是否把某个人关进监狱）也是一个推荐系统的例子，它与零售没有任何关

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

系，但是它将使用相同的模式匹配功能来检测一些更复杂的欺诈案例。

14.9 小结

在本章中，我们介绍了如何在推荐系统中使用诸如 Neo4j 之类的图数据库。我们没有讨论很多事情，这超出了本书的范围，但可能是真正的企业级推荐系统的一部分。然而，我们希望能够说明 Neo4j 的查询能力，它将为实时推荐系统提供大量的新机会，不需要预先计算，可以在接近实时的情况下使用。

下一章将使用一个案例来告诉你如何分析变化对过程或系统的影响，还将告诉你如何通过图表分析影响。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

第 15 章

◀ 案例——影响分析与模拟 ▶

在本章中，我们将学习如何将 Neo4j 图数据库用于另一个非常特殊的案例：**影响分析**。到目前为止，我们一直在讨论，如何使用 Neo4j 图数据库去处理特定的用例。

本章将学习以下内容：

- 剖析和理解影响分析系统。分成两个子主题：
  - 影响分析
  - 影响模拟
- 使用详细的演示数据集，将影响分析应用于业务流程管理用例。
- 使用详细的演示数据集，将影响模拟应用于成本计算用例。

下面我们马上开始吧！

15.1 影响分析系统解析

在本章中，我们先花点时间分析一下能够执行某些复杂操作的特定系统。搭建信息技术系统来处理这些操作，有时则恰恰相反。在这种情况下，分析和理解他们的业务以及市场对业务的影响会非常复杂。事实上，这样做意味着公司需要处理以下两件事情：

- 需要了解业务如何应对来自市场或其他方面的冲击。假设我们已经将相互关联的流程、人员和资源等业务建模，图数据库这个有趣的工具，能够让我们更直观地理解，当关系网络的一部分发生变化时是如何影响其他部分的。我们称之为核心影响分析用例。
- 需要能够模拟不同的潜在场景，并选择最优方案来达成具体目标。这与前文提到的场景是不一样的，不同之处在于，需要多次迭代前文提到的场景，来确定业务的最优终止状态需要达成何种目标。我们将这个称为影响模拟用例。

现在我们来更详细地研究这两个用例，并应用到具体的实例中。

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

15.1.1　影响分析在业务流程管理中的应用

在业务流程管理用例中，我们将探索如何使用图数据库技术来更好地理解不同业务流程及其关键资源的相互关联关系。这个用例在许多大型公司被称作业务流程管理，通常属于业务连续性管理的一部分。

显而易见：如果你有一个大公司，你的根本职责是时刻准备着处理突发事件（自然灾害、大规模停电、火灾、暴力恐怖袭击或任何其他重大破坏性事件）。为了达成目标，准备过程中至关重要的是了解我们要学习什么、Neo4j 图数据库可以做什么。

15.1.2　业务图建模

下面来看一个真实用例，为了便于理解，我们把很多业务概念组合为一张图。概念如下：

- **业务流程**：高层管理人员通常会考虑的高级业务流程。
- **流程**：组成业务流程的子流程。
- **恢复操作目标**：业务流程及其子流程将有一个最大恢复操作的时间范围，在这个时间范围内，异常事件会导致操作中止。恢复操作目标（RTO）的时间可以设置为从几分钟到几天的任何事情。
- **业务线**：流程将应用于一个或多个部门或业务线，这些组织部门能够使公司达成目标。
- **地点**：业务线位于特定的地点或建筑物。
- **应用程序**：使用特定的信息技术应用程序促使流程得以执行。

上面提到的概念都是相互关联的，如图 15-1 所示。

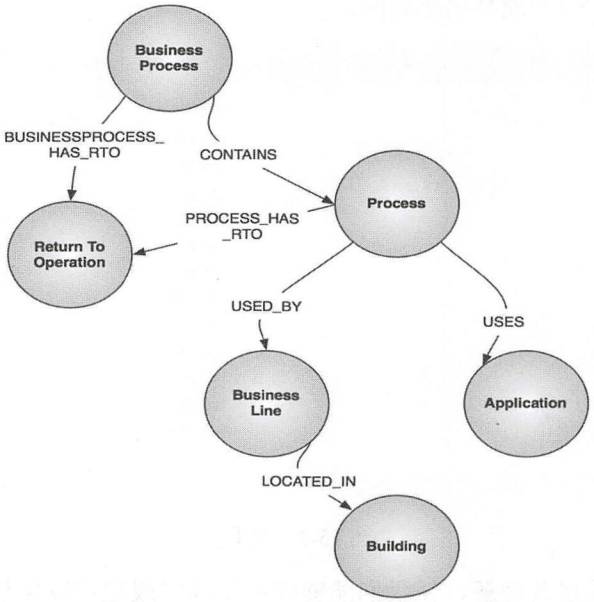


图 15-1　业务连续性管理模型

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

我们可以很容易地把样本数据集导入 Neo4j，通过简单的数据库查询，我们能够得到图 15-2 所示的信息。

```
CYPHER MATCH (a)-[r]->(b) WHERE labels(a) <> [] AND labels(b) <> [] WITH head(labels(a)) AS This, type(r) as To,
```

This	To	That	NrOfRels
Process	USES	Application	499
BusinessProcess	BUSINESSPROCESS_HAS_RTO	RTO	25
BusinessLine	LOCATED_IN	Building	264
RTO	PRECEDES	RTO	6
Process	PROCESS_HAS_RTO	RTO	421
Process	USED_BY	BusinessLine	448
BusinessProcess	CONTAINS	Process	325

✓ Returned 7 rows in 107 ms

图 15-2　BCM 数据内容

从中我们可以看到，样本数据集总共有 817 个节点和 1988 个关系，这足以让我们提出一些有趣的问题。

1. 应用程序用在什么地点

假设我们想要使用图模型来确定在我们公司的特定位置或建筑中使用哪些应用程序，执行下面的 Cypher 查询语句：

```
MATCH (app:Application)<-[:USES]-(proc:Process)- [:USED_BY]->(bl:BusinessLine)-
[:LOCATED_IN]->(b:Building)
RETURN DISTINCT app.name AS Application , b.name AS Building ORDER BY app.name ASC;
```

注意，因为没有起始节点，在图遍历时这是一个全图查询，执行时代价巨大。本例中的数据少，结果（见图 15-3）能够快速返回。

```
CYPHER MATCH (n:Application)<-[:USES]-(m:Process)-[:USED_BY]->(l:BusinessLine)-[:LOCATED_IN]->(b:Building) RETURN
```

Application	Building
Appl_1	Loc_186
Appl_1	Loc_32
Appl_1	Loc_87
Appl_1	Loc_62
Appl_1	Loc_205
Appl_1	Loc_40
Appl_1	Loc_154
Appl_1	Loc_51
Appl_1	Loc_25
Appl_1	Loc_20
Appl_1	Loc_22
Appl_1	Loc_158

✓ Returned 655 rows in 446 ms

图 15-3　结果

对业务的这种新发现及理解，使我们能够在应用程序或建筑发生任何事情时更好地采取行动。让我们继续深入了解。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

2. 如果 Appl_9 发生变化哪个地点有影响

我们把前面的查询局部变化一下，就能够很好地回答这个问题。应用程序在某些时候可能会遇到麻烦。我们来试试下面的查询：

```
MATCH (app:Application {name:"Appl_9"})<-[:USES]-(proc:Process)-[:USED_BY]->
(bl:BusinessLine)-[:LOCATED_IN]->(b:Building)
RETURN DISTINCT app,proc,bl,b;
```

由于我们只返回节点，并不包含节点的属性，Neo4j 前端浏览器立即返回一个我们需要的图展示页面（见图 15-4）。

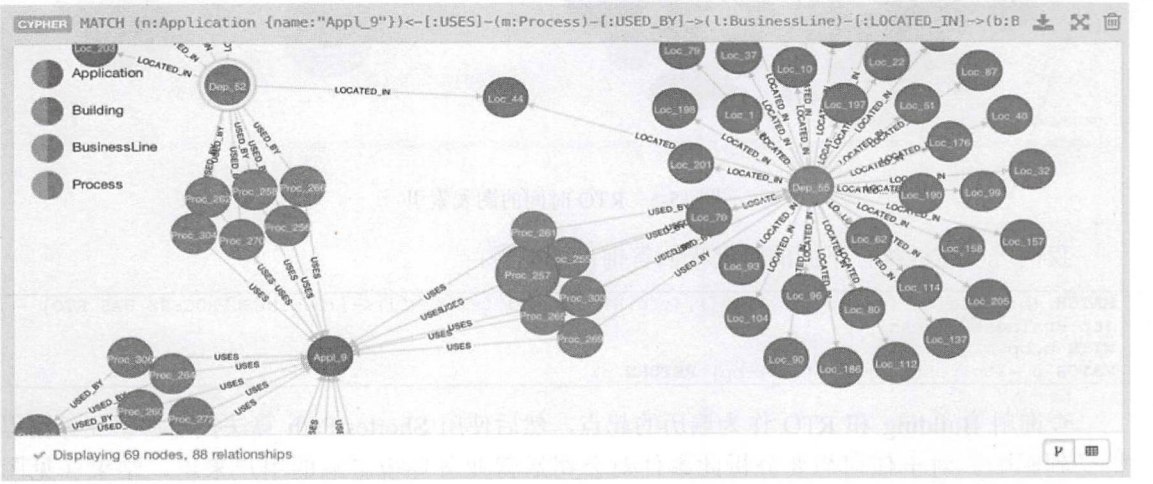


图 15-4　Appl_9 变化影响的建筑

诚然，这已经是对于用例非常好的解释，但是我们可以通过包含 RTO 信息来让它变得更复杂一些。让我们来尝试更多的查询。

3. 在位置 Loc_100 上，0~2 小时 RTO 受火灾影响的业务流程是什么

你也许已经注意到 RTO 特性中有一些独特的特征。许多人会认为这样的特征将成为（商业）过程的属性，但在我们的模型中，创建了单独的节点来表示这些特征，并把它们连接起来，形成一串 RTO 值：

- 0~2 小时 RTO 先于 2~4 小时。
- 2~4 小时 RTO 先于 4~24 小时。
- 4~24 小时 RTO 先于 1~7 天。
- 1~7 天 RTO 先于 7~14 天。
- 7~14 天 RTO 先于超过 14 天。

你可以在图 15-5 所示的浏览图中看到这些。这样做的好处是，我们现在可以非常容易地沿着 RTO 线找出与 RTO 类似或接近的目标。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

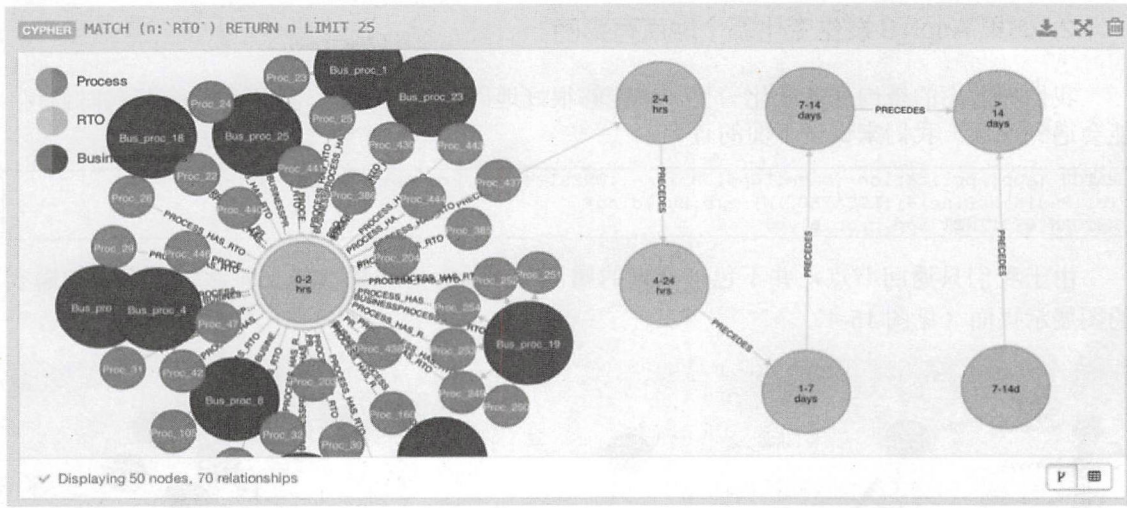


图 15-5　RTO 时间的图表索引

我们可以通过执行下面的查询来回答前面的问题：

```
MATCH (b:Building {name:"Loc_100"}),(rto:RTO {name:"0-2 hrs"})<-[:BUSINESSPROCESS_HAS_RTO] -
(bp:BusinessProcess)
WITH b,bp
MATCH p =ShortestPath(b-[*..3]-bp) RETURN p;
```

查询用 Building 和 RTO 作为遍历的起点，然后使用 ShortestPath 算法找出哪些业务流程将受到影响。对于任何想要分析此事件对公司运营业务环境影响的用户来说，结果（见图 15-6）一目了然。

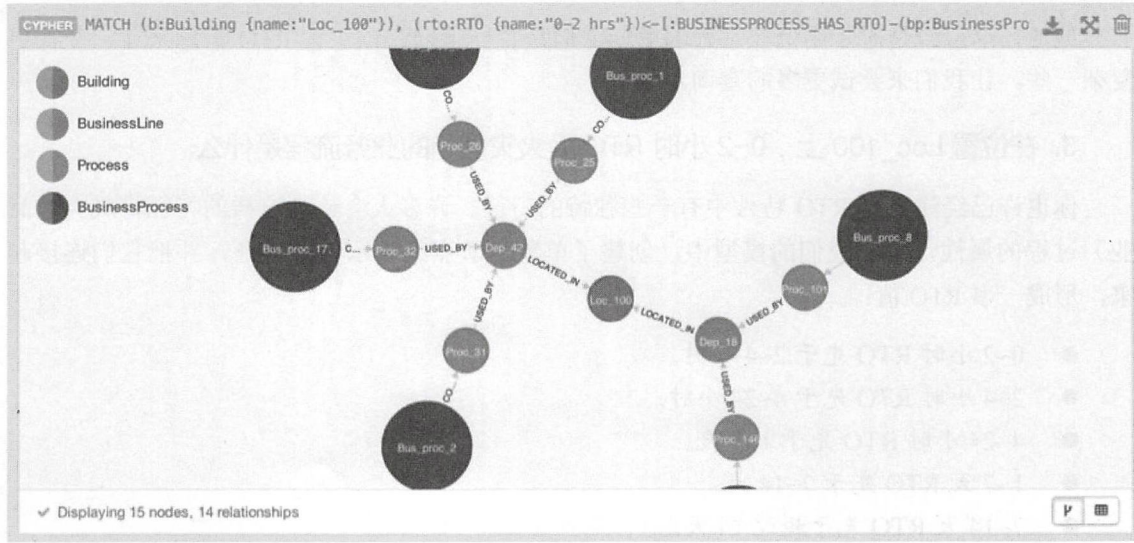


图 15-6　地点变化影响的 RTO 业务流程

通过这些例子，我们可以很好地了解 Neo4j 等图数据库如何帮助影响分析系统。接下来，将探索与这个领域相关的另一种场景：**影响模拟**。

15.2 成本计算环境中的影响模拟

正如前文所讨论的那样，本章中我们将讨论的第二个与影响有关的用例是模拟网络中其他网络对网络的某种变化的影响。基本问题是：如果网络发生某种变化，我们将会受到怎样的影响？如果我们改变图结构中的某些参数，最佳方案是什么？

本节中我们将使用的示例，你可能在其他用例中看到过这种结构，它使用了一种常见的数据结构：层次结构或树。在本节的学习过程中，将遍历树结构的不同部分，看看树的变化对其余部分的影响。

15.2.1 产品层次结构的图建模

我们正在研究的用例使用了产品层次结构的假设数据集。普遍存在于制造和建筑行业：

- 成本：若干组件组成。
- 产品：若干成本组成。
- 成本组：若干成本类型组成。
- 成本类型：若干成本子类型组成。
- 成本子类型：若干成本组成。

组件作为层次结构的最低部分，将具有特定的价格。在我的博客（http://blog.bruggen.com/2014/03/using-neo4j-to-manage-and-calculate.html）中，已展示如何方便地创建这样的样本数据集。该模型类似于图 15-7 所示的模型。

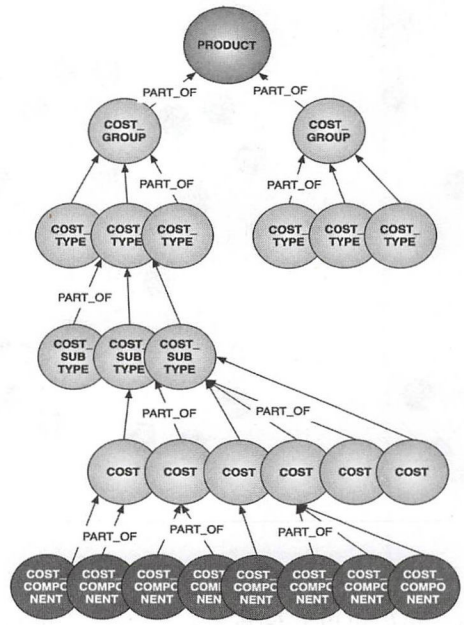


图 15-7　用于成本模拟的产品层次数据模型

在影响模拟的场景中，假设一个或多个组件价格变化时，如何快速有效地计算和重新计算产品的价格。

15.2.2 产品层次结构图的应用

先看如何用 Neo4j 构建产品层次结构图。将数据添加到图后，在 neo4j 数据库可以看到图 15-8 所示的信息。

```
CYPHER match (n) return labels(n) as Label, count(n) order by count(n) ASC;
```

Label	count(n)
PRODUCT	1
COST_GROUP	100
COST_TYPE	10000
COST_SUBTYPE	100000
COST	500000
COST_COMPONENT	1500000

✓ Returned 6 rows in 4096 ms

图 15-8 数据库信息

产品层次结构图总共约有 210 万个节点和相同数量的关系，产品层次非常巨大。在层次结构上做简单的查询，可以看到整个从上到下的结构，如图 15-9 所示。

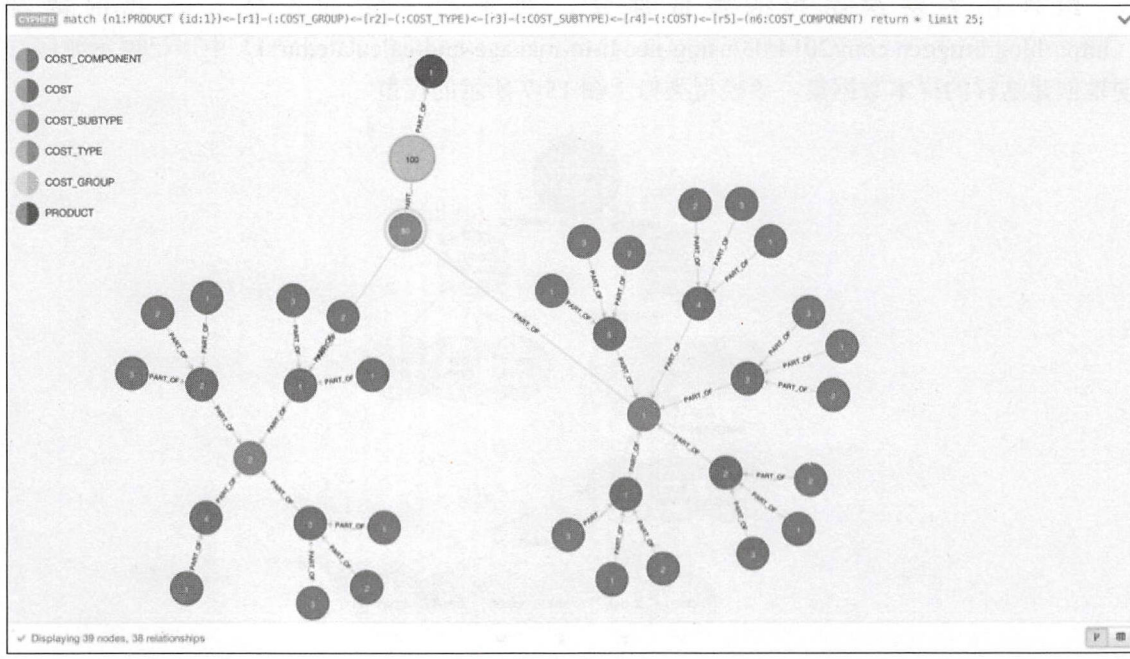


图 15-9 Neo4j 浏览器中的产品层次结构

现在已经有了层次结构，我们可以开始执行一些查询。我们要解决的关键问题之一是价格计算问题：如何遍历整个层次结构，并使用层次结构中存储的信息来重新计算产品的价

格？在关系系统中，这通常是非常困难和耗时的查询（事实上，这是一个五路连接操作）。那 Neo4j 会如何处理呢？

接下来，将采用两种不同的策略来计算产品层次结构中的产品价格。

- 第一个策略是遍历树，取树中价格信息的最低部分，然后乘以所有关系中的数量信息。当这个树完全扫描到顶端，我们将计算出产品的全部价格。
- 第二个策略是在树的每一层使用一些中间计算。我们将更新树的所有中间层次，并计算下层树的全部价格，因此，实际上只需要遍历树的很小一部分来重新计算价格。

让我们再来详细探讨一下。

1. 基于全树遍历的价格计算

在层次结构中，计算产品价格的第一个策略是遍历全树：

（1）从顶部的产品开始。

（2）向下遍历数据库中每个单一成本组件。

（3）返回所有价格的总和，作为每个成本部分的价格，与将成本部分连接到顶部产品的关系的数量的乘积计算。

该方法类似于以下查询：

```
MATCH (n1:PRODUCT {id:1})<-[r1]-(:COST_GROUP)<-[r2]-(:COST_TYPE)<-[r3]- (:COST_SUBTYPE)<-
[r4]-(:COST)<-[r5]-(n6:COST_COMPONENT)
RETURN sum(r1.quantity*r2.quantity*r3.quantity*r4.quantity*r5.quantity*n6.price) AS
CalculatedPrice
```

当执行这个查询时，需要相当长的时间，因为要遍历树的所有节点和关系以获得我们想要的结果。遍历拥有 210 万个节点和关系的树，我的笔记本电脑耗时约 47 秒，看图 15-10 中显示的结果。

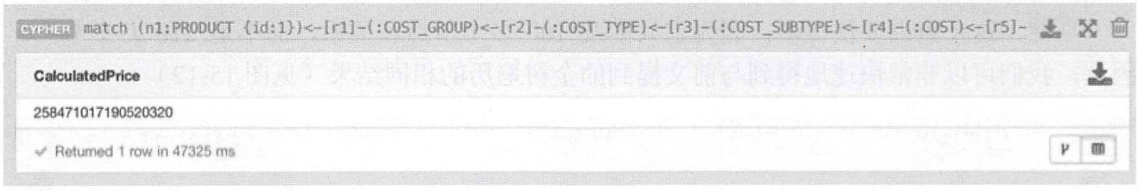


图 15-10　基于整个层次结构的价格计算

在关系系统里，任何执行过类似查询的人都将对查询结果印象深刻。通过优化层级管理策略，并在查询中使用中间价格信息，将大大提升查询效率。如果我们要执行影响模拟（本书这一部分的主题），我们期望能够快速计算出价格。现在我们再来看看。

2. 基于中间价格的价格计算

本例中使用的优化策略是在树的每个层级使用中间价格。这样，在计算价格时只需要遍历更小的子图；我们在做影响模拟时，仅仅需要遍历图中发生变化的部分（发生变化的节点/

关系，以及层次结构中的之前部分）。

为此，我们需要将中间价格信息添加到层次结构中。目前，只有最低的成本组件层拥有价格信息。此外，我们还需要计算各层级数据，可以执行以下查询：

```
MATCH (n5:COST)<-[r5]-(n6:COST_COMPONENT) WITH n5, sum(r5.quantity*n6.price) AS Sum
SET n5.price=Sum;
```

本质上，我们所做的就是从某个层级开始，向上遍历到上一层级，并且将上层的价格设定为下层所有产品的价格之和并乘以连接到上层关系的数量。

如前所述，我们需要对层次结构中每个层级都执行这种查询（见图 15-11）。

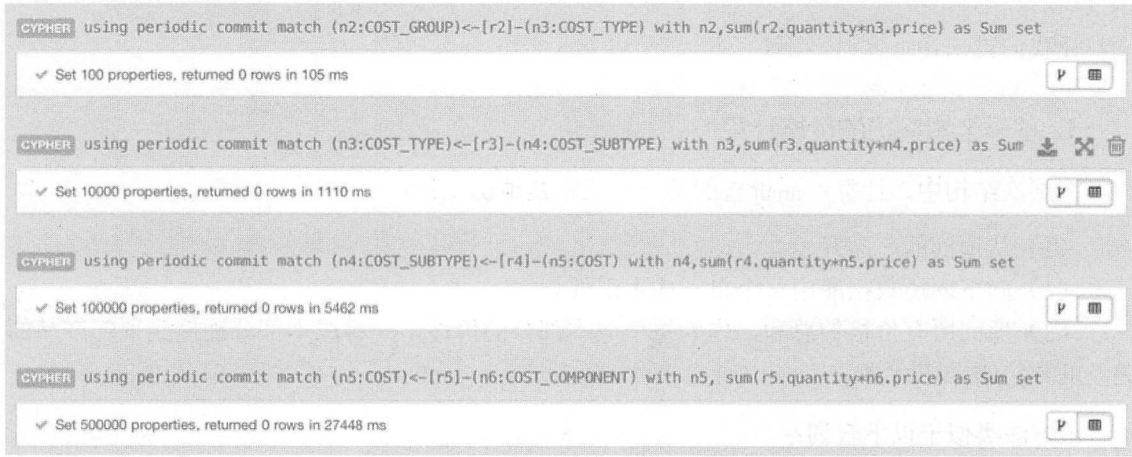


图 15-11　添加中间价格信息

大约 35 秒后，大功告成，这太不可思议了。

现在我们有了中间价格信息，计算产品的价格只需要遍历一个层级深度。我们使用类似于以下的查询：

```
MATCH (n1: PRODUCT {id: 1})< - [r1] - (n2: COST_GROUP) RETURN sum(r1.quantity* n2.price)
```

当然，这个查询要快得多，因为它只涉及 100 多个节点和关系，而不是 210 万个节点。因此，我们可以非常快速地得到与前文提到的全树遍历的相同结果（见图 15-12）。

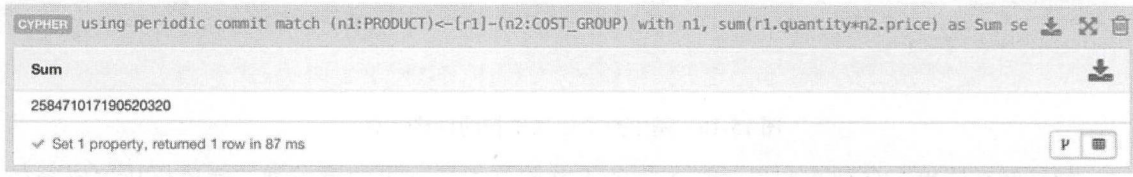


图 15-12　基于中间价格的价格计算

现在我们已经完成优化，可以使用该机制来进行影响模拟查询了。

3. 产品层次的影响模拟

我们最初的目标是影响模拟，通过模拟若干成本组件价格发生变化来分析产品上层结构

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

的价格影响。如果这种模拟足够高效，那么我们可能会频繁地对这些变化进行迭代，并将其用于根据产品的价格计算优化所有流程。

在之前简要介绍的优化策略中，我们可以很容易地改变成本组件的价格（位于层次结构的底部），当我们更新价格时，重新计算高于成本组件层级的中间价格（成本、成本子类型、成本类型、成本组、产品）。来看下面的查询（它由多个部分组成）：

Part	Query
Part 1	`MATCH (n6:COST_COMPONENT) WITH n6, n6.price AS OLDPRICE LIMIT 1` `SET n6.price = n6.price*10 WITH n6.price-OLDPRICE AS PRICEDIFF,n6`
Part 2	`MATCH n6-[r5:PART_OF]->(n5:COST)-[r4:PART_OF]->(n4:COST_SUBTYPE)-[r3:PART_OF]->` `(n3:COST_TYPE)-[r2:PART_OF]->(n2:COST_GROUP)-[r1:PART_OF]- (n1:PRODUCT)`

Part 3	`SET`
	`n5.price=n5.price+(PRICEDIFF*r5.quantity),`
	`n4.price=n4.price+(PRICEDIFF*r5.quantity*r4.quantity),`
	`n3.price=n3.price+(PRICEDIFF*r5.quantity*r4.quantity*r3.quantity),`
	`n2.price=n2.price+(PRICEDIFF*r5.quantity*r4.quantity*r3.quantity*r2.quantity),`
	`n1.price=n1.price+(PRICEDIFF*r5.quantity*r4.quantity*r3.quantity*r2.quantity`
	`*r1.quantity)`
	`RETURN PRICEDIFF, n1.price;`

- 第一部分（Part 1）：查找单个成本组件的价格，通过将其乘以 10 来更改它，然后将新价格和旧价格（PRICEDIFF）之间的差异传递给查询的下一部分。
- 第二部分（Part 2）：遍历至树层次结构的顶部，并查询出第一部分执行的更改操作影响的层次结构中的所有部分。
- 第三部分（Part 3）：使用第一部分和第二部分的信息，重新计算并设置中间层级的价格。在树的顶部（n1），将返回产品的新价格。

在数据集上执行该查询，得到图 15-13 所示的结果。

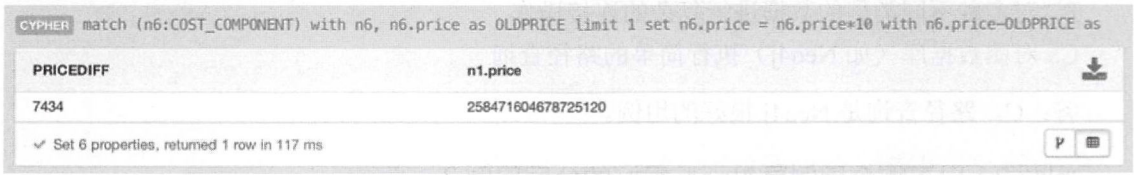


图 15-13　价格变化影响的价格重计算

重新计算 210 万件产品的价格，仅仅需要 117 毫秒，太不可思议了。让我们来看看如果改变多个成本组件的价格，这也是可行的。继续使用前面的查询，将 limit 1 更改为 limit 100，然后执行该查询，则会得到如图 15-14 所示的结果。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
CYPHER match (n6:COST_COMPONENT) with n6, n6.price as OLDPRICE limit 100 set n6.price = round(n6.price*1.05) with n6.price-
```

PRICEDIFF	n1.price
413	258471637316958720
45	258471639717409920
26	258471640744763520
16	258471648347180160
2	258471649187832000
42	258471650722935360
19	258471651103006800
30	258471657037455580
1	258471657044123520
35	258471673250873500
24	258471676066217500
11	258471680752283520

✓ Set 600 properties, returned 100 rows in 314 ms

图 15-14　基于 100 个价格变化影响的价格重计算

重新计算产品价格 314 毫秒就完成了，这仍然是一个非常棒的结果，为我们提供了模拟影响数据集的有效方法。

我们对影响模拟的讨论先告一段落。

15.3 问答

问题 1：影响分析通常需要做什么？

A. 对数据库大量的人工操作
B. 对多种不同类型的数据进行复杂的连接操作
C. 对图数据库（如 Neo4j）执行简单的路径查询

答：C。路径查询是 Neo4j 很好的用例。

问题 2：以下哪个用例是 Neo4j 很好的分层用例？

A. 计算超过一百万次不同购买交易的平均销售价格
B. 计算组和部门中用户目录树的访问控制
C. 计算两个城市之间的路网最短路径
D. 向老用户推荐新产品

答案：B。访问控制通常是很好的分层用例，目前市场上已经被多个分层结构的目录服务器验证过。然而，层次结构只是特定的图类型，通常也是 Neo4j 很好的用例。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

15.4 小结

我们在本章中了解到，在影响分析和模拟许多企业体系结构过程中，像 Neo4j 这类图数据库扮演着非常重要的角色。Neo4j 可应用于诸多领域，而我们只选择了两个特定的领域来说明这个用例。首先，我们研究了业务流程管理用例，分析和理解网络变化的潜在影响将是用户最感兴趣的。然后，我们又学习了影响模拟用例，在用例中，使用产品层次结构不断地模拟不同的影响场景来分析网络变化的结果是什么。

真心希望你通过本章学习，能够对本用例及其潜在应用场景有一个非常好的了解。

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

附 录

◀ 应用技巧介绍 ▶

这里列出一些提示和技巧，以便高效地使用 Neo4j。

重置密码

即使对我每天开发都用的机器，我可能都已经忘记了用户 Neo4j 的密码，因为在我的计算机上安装了好几个版本。

解决的办法是停止服务，然后进入 data/dbms 子目录，删除名为 auth 的文件。

检查其他主机

想知道在局域网中运行了哪些主机，可以启用 SSH 访问，我提供这个小脚本：

```
#!/bin/bash
port=22
for ordi in `nmap -sn 192.168.0.1-100 /25 | egrep "scan report" | awk
'{print $5}'`
do
echo Found IP $ordi on network.
echo ---------------------------------------
done
echo Now trying them for port $port , sorry for the timeout on
successful
connections
echo
for ordi in `nmap -sn 192.168.0.1-100 /25 | egrep "scan report" | awk
'{print $5}'`
do
echo Trying $ordi on port $port
cat < /dev/tcp/$ordi/$port
echo ---------------------------------------
done
```

实际环境下，可根据需要修改 IP 范围。UNIX 超级用户可能想知道为什么是/25 而不是/24。答案就是避免在操作循环中有.254 的 IP，因为它是路由器而不是主机。

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

获取 CSV 文件的第一行

得到一个 CSV 文件的第一行对于写 Cypher 查询的核心是非常有用的。如果我们能在命令行提示符下复制/粘贴其第一行，那为什么还要使用电子表格编辑器去做呢？如果文件很大以致需要较长时间打开，这将更有用。

在 Linux 机器上，尝试以下操作：

```
head -1 theBigHugeCSVFile.csv
```

如果我们想创建 Cypher 文件，尝试用常用的重定向操作：

```
head -1 theBigHugeCSVFile.csv > myFantasticImport.cyp
```

现在，我们可以用喜欢的文本编辑器打开 cyp 文件，并对字段分隔符进行替换，比如使用一个换行符（\n）、等号、我们想要用的变量名（比如 line）以及一个点号。

这样，我们就可以用一个可用的表单来填充所有字段：

```
=line.field1
=line.field2
=line.field3
```

使用最喜欢的文本编辑器，我们可以轻松地将前面的行转换成以下几行：

```
LOAD CSV WITH HEADERS FROM file:///theBigHugeCSVFile.csv' AS line
CREATE (a:SomeNodeLabel)
SET a.prop1=line.field1
SET a.prop2=line.field2
SET a.prop3=line.field3
```

这样，导入查询成为现实。（注意：输入的为字符串，需要时字段可通过 toFloat、toInteger 以及 toBoolean 函数来转换。）

启用树莓派（Raspberry Pi）上的 SSH

当我在 SD 卡上安装最新发行版的树莓派操作系统时，不幸的是，我没有显示器可接。这样太糟糕了，因为它无法自主运行常规设置。更糟糕的是，Pi 在运行，什么也不做，也无法访问。

幸运的是，通过启用已经安装的 SSH，远程访问就可以了。

当我们把指定 ISO 烧录到 SD 卡时，就会出现两个驱动器。一个是 Pi 的未来根文件系统，第二个名叫 boot，在那里我们必须创建一个名为 ssh 的空文件（没有扩展名）。

接下来要做的是插入一根网线，保持在线。至此可以通过 SSH 登录并运行 raspi-config。

早期版本的 Raspian 默认启用 SSH。也许我们使用的发行版有它。不管怎样，大家现在应该知道怎么做。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

Neo4j 的浏览器创建向导

当我们需要向用户显示数据但不会为其创建应用程序时，要考虑的一个选项是创建向导，使用本地托管以便用户只需单击查询即可获得结果。如果他们的权限是只读的就没有风险。

第一个显而易见的选项是查看入门指南并查看源代码是如何构造的。第二个选项是看neo4j guides 开源项目，可在这里查看：https://github.com/neo4j-contrib/neo4j-guides。

这个工程实现将 asciidoc 格式文档转换成 HTML 指南。

这是入门指南：

```
== Learning Neo4j
:author: Jerome BATON
:twitter: wadael
=== Guides
Guides are very useful to quickly propose read-only queries to users.
See the arrows, I wrote no distinct source for that. Click it please
=== Add pictures
You can add pictures with this code
[source, text]
----
image::https://neo4j.com/wp-
content/themes/neo4jweb/assets/images/neo4j-log
o-2015.png[height=150,float=right]
----
image::https://neo4j.com/wp-
content/themes/neo4jweb/assets/images/neo4j-log
o-2015.png[height=400, float=right]
Neat.
== Queries
=== How many nodes ?
If you click on the query below, it will be copied in the input. You
will
just have to run it.
[source, cypher]
----
MATCH (n)
RETURN count(n)
----
```

步骤如下：

步骤 01 使用 git 工具克隆这个工程。

步骤 02 将代码粘贴到工程目录下的 learning.adoc 里。

步骤 03 执行./ run.sh ./ learning.adoc。

步骤 04 用 apt-get install asciidoctor 和 gem install tilt 解决依赖问题。（可选）

步骤 05 在 neo4j.conf 文件的结尾添加 browser.remote_content_hostname_whitelist=*。（警告：它允许来自互联网的访问。）

步骤 06 执行 Python 脚本 http-server.py。

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

步骤 07　在 Neo4j 的浏览器里，执行 play http://localhost:8001/html/learning.html。

步骤 08　测试我们的入门指南案例。

步骤 09　将其部署到生产服务器上。

数据备份和恢复

这取决于我们所使用的版本，其备份和还原数据的方式将有所不同。

社区版

要备份社区版本服务器的数据，只有冷备（脱机）是可行的。

很基本的，但并不是说 DIY：先停止服务，然后复制整个$NEO_HOME/data 目录到安全存储位置，再重启服务。

恢复备份数据，先停止服务，用备份替换$NEO_HOME/data 目录，再重启服务。

另一个方法是使用 neo4j-admin 实用程序，安装服务时放在$NEO_HOME/bin 下：

```
neo4j-admin dump --to=file --database=optionnalDBName
```

to 参数是文件的路径。database 参数是可选的，仅在拥有多个数据库且想要转储不是 graph.db 的数据库时才需要。

为了恢复数据的转储文件，命令如下：

```
neo4j-admin load from=file
```

企业版

企业版提供了一个增强版的实用程序：neo4j-admin，位于$NEO_HOME/bin 下。这个版本有两个函数：backup 和 restore。

这个允许在线备份，需要一些配置；我们需要注释$NEO_HOME/conf/neo4j.conf 里粗体显示的两行：

```
# Enable online backups to be taken from this database.
dbms.backup.enabled=true
# By default the backup service will only listen on localhost.
# To enable remote backups you will have to bind to an external
# network interface (e.g. 0.0.0.0 for all interfaces).
dbms.backup.address=0.0.0.0:6362
```

工具集

接下来让我们看几个工具。

Cypher-shell

Cypher Shell 是一个命令行工具，在 Neo4j 的默认安装里自带。

我们可以在$NEO_HOME/bin 目录下找到它。如前所述，它允许我们在 Neo4j 服务器上

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

运行交互式 Cypher 查询，就像在 Neo4j 浏览器里一样：

```
jerome@nitendet ~/Tools/neo/neo4j-enterprise-3.2.3/bin $ cypher-shell
username: uadmin
password: ******
Connected to Neo4j 3.2.3 at bolt://localhost:7687 as user uadmin.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j> MATCH (n)
       RETURN count(n);
+----------+
| count(n) |
+----------+
| 8        |
+----------+

1 row available after 1 ms, consumed after another 0 ms
neo4j> :quit

Bye!
jerome@nitendet ~/Tools/neo/neo4j-enterprise-3.2.3/bin $
```

也可以用类 UNIX 方式使用它，以一个文件结尾。假如我们有一个名叫 count.cy 的文件，含有如下查询：

```
MATCH (n)
RETURN count(n);
```

我们可以这样使用 Cypher-shell（系统管理员模式）：

```
jerome@nitendet ~/Tools/neo/neo4j-enterprise-3.2.3/bin
Fichier Édition Affichage Rechercher Terminal Aide
jerome@nitendet ~/Tools/neo/neo4j-enterprise-3.2.3/bin $ more count.cy
MATCH (n)
RETURN count(n);
jerome@nitendet ~/Tools/neo/neo4j-enterprise-3.2.3/bin $ cypher-shell -u uadmin
-p uadmin --debug --format plain < count.cy
count(n)
8
```

数据集成工具

Talend（www.talend.com）是一个开源软件供应商，提供数据集成、数据管理、企业应用集成、大数据的软件和服务，有一个用于各种集成服务的可扩展、可视、强大的开发环境，可在一个工作流设计器风格的环境里将各组件连接在一起。

MuleSoft（www.mulesoft.com）总部设在美国加利福尼亚州的旧金山，提供了一个集成平台，无论在云端还是本地部署，均可连接任何应用程序、数据源或 API。就像 Talend 一样，它提供了一个可视环境，可以整合在带有 Neo4j 图数据库管理系统的混合持久化架构里的其他数据源等。为了实现这个，MuleSoft 提供文档良好的 Neo4j 连接器，允许我们非常容易地整合数据。

非卖品，仅供非商业用途或交流学习使用

严禁网络传播本PDF，违者责任自负！

建模工具

图数据库的建模与以往一样重要。即使默认没有检查我们数据模型的外部模式，仍然有一个明确而重要的建模必要。如何定义一个我们计划存储在 Neo4j 中的数据结构呢？需要用一个明确的和可以理解的方式记录。本节将为图数据库建模开发人员介绍两个常用的工具。

Arrows

最初，Neo 技术公司 Alistair Jones（www.apcjones.com）把它作为图形文档化和可视化方面的项目，Arrows 工具现在能够生成很方便和有用的图数据库建模文档。该工具在网上（www.apcjones.com/arrows）可以找到，且提供很基本但先进的绘图能力。

关于 Arrows 模型最美妙的事情是使 Neo4j 属性图模型映射到绘图画布。它的自动缩放和对齐功能也非常有用。

Arrows 工具集是一个公开可用的开放源代码项目：https://github.com/apcj/arrows。

OmniGraffle

OmniGraffle（http://www.omnigroup.com.omnigraffle/）是一个由 Omni Group（http://www.omnigroup.com）开发的图表应用软件。OmniGraffle 仅运行在 Mac OS X 和 iPad 上。它可用于创建图表、流程图、组织结构图、插图以及图数据库模型。它是拖曳式所见即所得的界面。模板（可拖曳的图形组）是 OmniGraffle 的扩展，并且用户可以创建自己的模板，例如，简单的图形编辑的要求。在许多方面，OmniGraffle 类似微软的 Visio。

社区工程

下面有几个明确的工程值得一看：

- https://github.com/cleishm/libcypher-parser：libcypher-parser 是一个分析库及验证（lint）工具，用于 Cypher——图查询语言。
- Structr (https://structr.org)：Struct 是一个应用程序生成器，做自带的样例程序，看看视频，将会有惊喜。
- https://neo4j-client.net/：一个命令行客户端。
- https://github.com/anvaka/ngraph.path：一个 JS 库，用在任意图里的快速路径发现。
- 还有一个值得我们花时间的可视化库，就是 Popoto.js（http://www.popotojs.com/）。它是用 D3.js 创建的。
- https://github.com/neo4j-contrib/neo4j-graph-algorithms：Neo4j 更高效的图算法集。

在线文档

Neo4j 公司提供了大量的产品文档，参见 https://neo4j.com/docs/。

YouTube 频道——https://www.youtube.com/c/neo4j 提供了图连接情况的视频。

在这里可以找到 APOC 的使用说明：https://neo4j-contrib.github.io/neo4j-apoc-procedures/。

非卖品，仅供非商业用途或交流学习使用　　严禁网络传播本PDF，违者责任自负！

社区

作为开源用户，我们有时寻求帮助，有时提供帮助。StackOverflow 上有一个 Neo4j 的标签：https://stackoverflow.com/questions/tagged/neo4j，针对所有的 Neo4j 问题。

还有几个 Slack 频道，参见 https://neo4j.com/developer/slack/。

如果大家使用 Twitter，可以关注@neo4j、@mesirii、@emileifrem、@rvanbruggen、@craigtaverner、@Aethelraed、@darthvader42、@tb_tomaz、@lyonwj、@maxdemarzi、@markhneedham、@BarrasaDV、@ryguyrg、@andres_taylor、@iansrobinson、@apcj、@jimwebber（以及或许也有 @wadael）。

图连接大会和当地的聚会是一种很好的途径，可以去会见业内同仁，交流技巧，并获得工作机会。

更多谚语

不太经意地合上这本书，但无论如何还是智慧的！下面给出一些谚语。

- 垃圾进！垃圾出！也就是说，如果在导入数据时不尽最大的注意，作为查询结果我们还是得到这些应得的数据。（正如 Lili 所说）
- 注意处理的空值、不要把数值作为字符串导入，并在需要的时候使用布尔值。
- 当把纬度和经度弄错时，一切都错了！
- 在书的封面上写上孩子的名字会使家庭幸福！
- 学习的乐趣呼唤教学的乐趣！
- 作者喜欢反馈！

非卖品，仅供非商业用途或交流学习使用

Neo4j 3.x 入门经典

Neo4j是一个流行的图数据库，用它可以轻松地遍历巨量相互连接的数据。本书旨在让您快速地入门Neo4j。

本书以图论的简单介绍作为开始，向您展示了使用图数据库的优势以及图数据库的建模技术。您将获得常用的实战经验，以及如何使用Neo4j的Cypher查询语言来更新图数据的技能。

此外，您还会学习到如何使用APOC中的过程以及您自己的过程来扩展Neo4j客户端的功能，开启集成、算法分析和其他高级的空间操作。

在本书中，您将了解到Neo4j的最新特性，如扩展性、性能优化、可视化、数据重构技术、安全增强等。在本书的最后，您将在一些现实世界案例的帮助下，获得设计和实现现代应用所需的技能，这些应用将从图数据中揭示其在商业上的能力。

通过本书你将会：

- 理解图论、数据库的基本原理以及Neo4j相比传统数据库的一些优势
- 安装Neo4j、数据建模和最常见的图数据遍历实践
- 学习Cypher查询语言和定制的过程来分析与驱动有意思的数据呈现
- 在APOC库的过程帮助下提高操作图数据的技术
- 使用Neo4j高级扩展和插件来做性能调优
- 理解大规模部署下Neo4j新的安全特性和集群架构

清华社官方微信号

扫 我 有 惊 喜

ISBN 978-7-302-51918-8

9 787302 519188 >
定价：69.00元